



Language Reference Manual

EASEL Version 1.1
for OS/2
Extended Edition

IBM PROGRAM LICENSE AGREEMENT

BEFORE OPENING THIS PACKAGE, YOU SHOULD CAREFULLY READ THE FOLLOWING TERMS AND CONDITIONS. OPENING THIS PACKAGE INDICATES YOUR ACCEPTANCE OF THESE TERMS AND CONDITIONS. IF YOU DO NOT AGREE WITH THEM YOU SHOULD PROMPTLY RETURN THE PACKAGE UNOPENED AND YOUR MONEY WILL BE REFUNDED.

This is a license agreement and not an agreement for sale. IBM owns, or has licensed from the owner, copyrights in the Program. You obtain no rights other than the license granted you by this Agreement. Title to the enclosed copy of the Program, and any copy made from it, is retained by IBM. IBM licenses your use of the Program in the United States and Puerto Rico. You assume all responsibility for the selection of the Program to achieve your intended results and for the installation of, use of, and results obtained from, the Program.

The Section in the enclosed documentation entitled "License Information" contains additional information concerning the Program and any related Program Services.

LICENSE

You may:

1. use the Program on only one machine at any one time, unless permission to use it on more than one machine at any one time is granted in the License Information (Authorized Use);
2. make a copy of the Program for backup or modification purposes only in support of your Authorized Use. However, Programs marked "Copy Protected" limit copying;
3. modify the Program and/or merge it into another program only in support of your Authorized Use; and
4. transfer possession of copies of the Program to another party by transferring this copy of the IBM Program License Agreement, the License Information, and all other documentation along with at least one complete, unaltered copy of the Program. You must, at the same

time, either transfer to such other party or destroy all your other copies of the Program, including modified copies or portions of the Program merged into other programs. Such transfer of possession terminates your license from IBM. Such other party shall be licensed, under the terms of this Agreement, upon acceptance of this Agreement by its initial use of the Program.

You shall reproduce and include the copyright notice(s) on all such copies of the Program, in whole or in part.

You shall not:

1. use, copy, modify, merge, or transfer copies of the Program except as provided in this Agreement;
2. reverse assemble or reverse compile the Program; and/or
3. sublicense, rent, lease, or assign the Program or any copy thereof.

LIMITED WARRANTY

Warranty details and limitations are described in the Statement of Limited Warranty which is available upon request from IBM, its Authorized Dealer or its approved supplier and is also contained in the License Information. IBM provides a three-month limited warranty on the media for all Programs. For selected Programs, as indicated on the outside of the package, a limited warranty on the Program is available. The applicable Warranty Period is measured from the date of delivery to the original user as evidenced by a receipt.

Certain Programs, as indicated on the outside of the package, are not warranted and are provided "AS IS."

Continued on inside back cover.



Language Reference Manual

EASEL Version 1.1
for OS/2
Extended Edition

Second Edition (May 1990)

This publication could include technical inaccuracies or typographical errors. Changes are made periodically to the information herein; these changes will be incorporated in new editions of this publication.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent program may be used instead.

Products and publications are not stocked at the address given below. Requests for copies of this product and for technical information about the system should be made to your local IBM representative.

A form for reader's comments is provided at the back of this publication. If the form has been removed, address your comments to IBM Corporation, Software Development, Dept. 877, 10401 Fernwood Road, Bethesda, Maryland 20817. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1990. All rights reserved.

© Copyright Interactive Images, Inc. 1989, 1990.

Note to US Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

IBM and OS/2 are registered trademarks of International Business Machines Corporation.

EASEL is a registered trademark of Interactive Images, Inc.

Preface

This publication provides a technical reference of the language statements and functions used with IBM EASEL® Version 1.1 for OS/2® Extended Edition for developing graphical user interface applications.

Throughout this publication IBM EASEL Version 1.1 for OS/2 Extended Edition is referred to as EASEL OS/2 EE.

This publication is organized alphabetically within each of the following chapters:

Chapter 1, Language Elements

Chapter 2, Functions and Subroutines

Chapter 3, Environment

Chapter 4, Include Files

For more information about a particular language statement or function, see the *IBM EASEL Version 1.1 for OS/2 Extended Edition Developer's Guide*.

EASEL OS/2 EE is developed by Interactive Images, Inc. of Woburn, MA.

IBM and OS/2 are registered trademarks of International Business Machines Corporation.

EASEL is a registered trademark of Interactive Images, Inc.

Notation Conventions

The syntax models of EASEL OS/2 EE language statements and functions presented in this publication use the conventions shown below.

Convention	Meaning	Examples
<code>^</code>	Indicates a control character.	<code>^D</code>
bold lowercase words	EASEL OS/2 EE statement name or keyword. You must specify exactly as shown.	region selected line is checked
bold symbols	You must specify exactly as shown.	<code>()</code> <code>[]</code> <code>:</code>
UPPERCASE WORDS	Value or identifier you specify. You must substitute your own identifiers or values.	<code>OBJECT_NAME</code> <code>LOOP_NAME</code> <code>LINE_NO</code>
Square brackets <code>[]</code>	Optional entry. You can optionally specify the entry in brackets.	<code>[primary]</code> <code>[column]</code>
	Multiple choice optional entry. You can select one or none of the entries in the brackets. Choices are separated by a vertical bar (<code> </code>).	<code>[guarded resumable]</code>
	When used to specify elements and dimensions in array definitions and references, you must specify exactly as shown.	<code>string ARRAY[N]</code>
Underlined keywords	The default. Does not need to be specified.	<code>[<u>enabled</u> disabled]</code>

Convention	Meaning	Examples
Braces { }	Required entry. You must select one of the entries in the braces. Choices are separated by a vertical bar ().	{ keyboard APPLICATION }
Ellipses ... (horizontal)	Optional repetition. You can repeat the preceding bracketed entry any number of times. When used without a preceding bracketed entry, indicates an omitted portion of syntax.	[ACTION_STATEMENT]... [in PARENT_NAME2]... response to border
: (vertical)	Omitted code. Portions of the EASEL OS/2 EE code have been omitted.	begin Block_Name : end Block_Name

Identifiers and Color Values

In all syntax models, except compile-time definitions, an *identifier* can be replaced with a string value representing the identifier. Compile-time definitions are those definitions not created during runtime with the **add** action statement.

In all syntax models, a compile-time definition *identifier* can be replaced with a compile-time string value representing the identifier.

In all syntax models, COLOR is a string value representing a color keyword, an integer value representing a color number, or a color keyword.

In all syntax models, the keyword **color** preceding COLOR is only optional if COLOR is specified as a color keyword, without quotation marks.

Contents

Chapter 1. Language Elements	1-1
Overview	1-2
action Action Statement	1-3
action bar Attribute Definition	1-4
action bar is Template Definition	1-5
action is Action Routine Definition	1-7
activate Action Statement	1-8
add Action Statement	1-10
add item to Action Statement	1-12
add to Action Statement	1-14
add to class Action Statement	1-17
add to item class Action Statement	1-18
ancestry Response Inquiry Built-in Function	1-19
ancestry of Object Inquiry Built-in Function	1-20
append Action Statement	1-21
application Environment Declaration	1-22
array Definition and Reference	1-23
background at Object Inquiry Built-in Function	1-26
background of Object Inquiry Built-in Function	1-27
begin BLOCK Action Statement	1-28
blank block Drawing Statement	1-31
block Definition (for Text)	1-32
border Attribute Definition	1-34
border of Object Inquiry Built-in Function	1-36
bottom [of] Object Inquiry Built-in Function	1-37
box Drawing Statement	1-38
Built-in Functions	1-39
Response Inquiry Built-in Functions	1-40
Object Inquiry Built-in Functions	1-40
Item Inquiry Built-in Functions	1-43
Action Inquiry Built-in Functions	1-43
Special Inquiry Built-in Functions	1-43
button Definition	1-45
call Action Statement	1-47
change Action Statement	1-48
change action bar Action Statement	1-50
change position Action Statement	1-51
change priority Action Statement	1-53
change size Action Statement	1-54
change text Action Statement	1-55
change to program Action Statement	1-57
change window position Action Statement	1-59
change window size Action Statement	1-61
Character Set	1-63
check and uncheck Action Statements	1-64
check box Definition	1-66

checked Object/Item Inquiry Built-in Function	1-68
checked in group Object Inquiry Built-in Function	1-69
choice Definition	1-70
circle Drawing Statement	1-73
class Definition	1-74
clear Action Statement	1-76
clipboard Special Built-in Function	1-78
close Action Statement	1-79
color Keyword	1-80
column size of Object Inquiry Built-in Function	1-83
combination box Definition	1-84
Comments	1-87
constant Definition	1-88
copy Action Statement	1-89
date Special Inquiry Built-in Function	1-91
deemphasize Action Statement	1-92
delete Action Statement	1-93
delete action bar from Action Statement	1-94
delete from class Action Statement	1-95
delete from item class Action Statement	1-96
delete item Action Statement	1-97
deselect Action Statement	1-98
dialog box Definition	1-99
Dialog Control Objects	1-101
dialog region Definition	1-102
disable Action Statement	1-106
disabled Attribute Definitions	1-107
draw Drawing Statement	1-108
draw arc Drawing Statement	1-110
dropdown list Definition	1-113
ellipse Drawing Statement	1-116
emphasize Action Statement	1-118
enable and disable Action Statements	1-119
enabled and disabled Attribute Definitions	1-121
enabled Item Inquiry Built-in Function	1-123
entry field Definition	1-124
errorlevel Action Inquiry Built-in Function	1-126
Escape Sequences	1-127
eventnumber Response Inquiry Built-in Function	1-128
eventparam Response Inquiry Built-in Function	1-129
exists Object Inquiry Built-in Function	1-130
exit Action Statement	1-131
Expressions, Operators, and Operands	1-132
extract from Action Statement	1-134
fileid Definition	1-138
find Action Statement	1-139
font Reference	1-140
font is Font Definition	1-143
for each member loop Action Statement	1-146
for each selected line loop Action Statement	1-148

for loop Action Statement	1-150
foreground at and background at Object Inquiry Built-in Functions	1-152
foreground of Object Inquiry Built-in Function	1-154
found Action Inquiry Built-in Function	1-155
Frame Attributes	1-156
freysize Special Inquiry Built-in Function	1-157
function Declaration	1-158
graphical region Definition	1-159
group box Definition	1-164
handle of Object Inquiry Built-in Function	1-166
highlight Action Statement	1-167
Identifiers	1-169
if Conditional Action Statement	1-170
image region Definition	1-172
imagemap Definition	1-178
in Keyword	1-179
include Reference	1-181
input Response Inquiry Built-in Function	1-183
insert Drawing Statement	1-184
invisible Attribute Definitions	1-187
invoke Action Statement	1-188
ioerror Action Inquiry Built-in Function	1-189
is checked Object/Item Inquiry Built-in Function	1-190
is enabled Item Inquiry Built-in Function	1-192
item Reference	1-193
item class Definition	1-194
key Definition	1-196
Keyboard Mapping Table	1-198
Keywords	1-202
leave block Action Statement	1-203
leave loop Action Statement	1-205
left [of] and right [of] Object Inquiry Built-in Functions	1-206
length of Special Inquiry Built-in Function	1-208
line size of Object Inquiry Built-in Function	1-209
list box Definition	1-210
make COLOR Action Statement	1-212
make block and make segment COLOR Action Statements	1-213
make border COLOR Action Statement	1-215
make border invisible and make border visible Action Statements	1-216
make border visible Action Statement	1-217
make cursor Action Statement	1-218
make invisible and make visible Action Statements	1-219
make parameter Action Statement	1-220
make point disposition Action Statement	1-221
make segment COLOR Action Statement	1-222
make string disposition Action Statement	1-223
make visible Action Statement	1-224
marker Special Built-in Function	1-225
member Special Inquiry Built-in Function	1-226
members of Special Inquiry Built-in Function	1-227

module Reference	1-228
move Drawing Statement	1-229
move arc Drawing Statement	1-231
multiline entry field Definition	1-234
object Response Inquiry Built-in Function	1-237
Object Types and Ancestry	1-238
open and close Action Statement	1-239
overwrite Drawing Statement	1-240
parameter Response Inquiry Built-in Function	1-242
parameter is Attribute Definition	1-243
parameter of Object Inquiry Built-in Function	1-244
pattern Pattern Reference	1-245
pattern is Pattern Definition	1-248
plot Action Statement	1-249
polygon Drawing Statement	1-251
position Keyword	1-252
precision Environment Declaration	1-253
priority is Attribute Definition	1-254
priority of Object Inquiry Built-in Function	1-255
pulldown Definition	1-256
push button Definition	1-258
radio button Definition	1-261
read Action Statement	1-263
record Definition	1-266
refresh Action Statement	1-269
remove Action Statement	1-270
resize Action Statement	1-272
response to OBJECT Response Definition	1-274
response to char and response to line Response Definitions	1-279
response to interval Response Definition	1-283
response to item Response Definition	1-284
response to line Response Definition	1-286
response to low memory Response Definition	1-287
response to start Response Definition	1-289
response to stimulus Response Definition	1-290
response to termination Response Definition	1-292
response to timeout Response Definition	1-293
right [of] Object Inquiry Built-in Function	1-294
save program as Action Statement	1-295
screen size Environment Declaration	1-297
segment Definition	1-299
select and deselect Action Statements	1-301
selectability of Object Inquiry Built-in Function	1-302
selected line Special Inquiry Built-in Function	1-303
selected line from Object Inquiry Built-in Function	1-304
send Action Statement	1-305
sense region Definition	1-307
separator Definition	1-310
set low memory threshold Action Statement	1-311
set pointer to Action Statement	1-312

shape Drawing Statement	1-313
squeeze memory Action Statement	1-316
start Action Statement	1-317
static text Definition	1-322
stimulus Declaration	1-324
stop Action Statement	1-325
subroutine Declaration	1-326
subroutine is Definition	1-328
switch Action Statement	1-330
text Drawing Statement	1-332
text of Object Inquiry Built-in Function	1-334
text of item Item Inquiry Built-in Function	1-335
textual Object Inquiry Built-in Function	1-336
textual region Definition	1-338
time Special Inquiry Built-in Function	1-343
top [of] and bottom [of] Object Inquiry Built-in Functions	1-344
turn trace Action Statement	1-346
Type Conversions	1-347
uncheck Action Statement	1-349
Values	1-350
variable Definition	1-352
visibility of Object Inquiry Built-in Function	1-354
visible and invisible Attribute Definitions	1-355
wait Action Statement	1-357
wedge Drawing Statement	1-359
while loop Action Statement	1-361
window xposition [of] and window yposition [of] Object Inquiry Built-in Functions	1-364
window xsize [of] and window ysize [of] Object Inquiry Built-in Functions	1-366
window yposition [of] Object Inquiry Built-in Function	1-367
window ysize [of] Object Inquiry Built-in Function	1-368
write Action Statement	1-369
xcoord and ycoord Response Inquiry Built-in Functions	1-371
xcursor [of] and ycursor [of] Object Inquiry Built-in Functions	1-372
xmiddle [of] and ymiddle [of] Object Inquiry Built-in Functions	1-373
xposition [of] and yposition [of] Object Inquiry Built-in Functions	1-374
xsize [of] and ysize [of] Object Inquiry Built-in Functions	1-375
ycoord Response Inquiry Built-in Function	1-376
ycursor [of] Object Inquiry Built-in Function	1-377
ymiddle [of] Object Inquiry Built-in Function	1-378
yposition [of] Object Inquiry Built-in Function	1-379
ysize [of] Object Inquiry Built-in Function	1-380
Chapter 2. Functions and Subroutines	2-1
Overview	2-2
Date Library	2-2
EASEL OS/2 EE Library	2-2
File I/O Library	2-2
Math Library	2-3

Message Library	2-3
ABS() Function	2-4
ARCCOS() Function	2-5
ARCSIN() Function	2-6
ARCTAN() Function	2-7
ARCTAN2() Function	2-8
CEIL() Function	2-9
CloseFile() Subroutine	2-10
COS() Function	2-11
DaySpan() Function	2-12
E() Function	2-13
EslQueryEgaColor() Function	2-14
EslQueryFocus() Function	2-15
EslSetEgaColor() Function	2-16
EXP() Function	2-18
FLOOR() Function	2-19
GetError() Subroutine	2-20
LN() Function	2-21
LocalDate() Function	2-22
LocalTime() Function	2-23
LOG() Function	2-24
MAX() Subroutine	2-25
MAXINT() Subroutine	2-26
MEAN() Subroutine	2-27
MEANINT() Subroutine	2-28
MEDIAN() Subroutine	2-29
MEDIANINT() Subroutine	2-30
MIN() Subroutine	2-31
MININT() Subroutine	2-32
MOD() Function	2-33
OpenFile() Subroutine	2-34
PI() Function	2-36
POWER() Function	2-37
ReadLineNumber() Subroutine	2-38
ReadNext() Subroutine	2-39
ReadNextRecord() Subroutine	2-40
ReadRecordAtLine() Subroutine	2-41
ReplyToMessage() Function	2-42
SetBufferSize() Subroutine	2-44
SetIndexSize() Subroutine	2-45
SetTabSize() Subroutine	2-46
SIN() Function	2-47
STDDEV() Subroutine	2-48
STDDEVINT() Subroutine	2-49
SQRT() Function	2-50
SUM() Subroutine	2-51
SUMINT() Subroutine	2-52
TAN() Function	2-53
Validdate() Function	2-54
VARIANCE() Subroutine	2-55

VARIANCEINT() Subroutine	2-56
WeekDay() Function	2-57
WriteRecord() Subroutine	2-58
WriteString() Subroutine	2-59
 Chapter 3. Environment Files	 3-1
Overview	3-2
COMPILE.CMD OS/2 Command File	3-3
CONFIG.ESL File	3-6
EASEL.CMD OS/2 Command File	3-10
 Chapter 4. Include Files	 4-1
Overview	4-2
accel.inc File	4-3
datelib.inc File	4-4
egacolor.inc File	4-5
eslappc.inc File	4-6
eslaudio.inc File	4-12
esllib.inc File	4-14
fileio.inc File	4-15
mathlib.inc File	4-16
message.inc File	4-18
pmptr.inc File	4-20
 Index	 X-1

Chapter 1. Language Elements

Overview

This chapter describes the syntax of the IBM EASEL® Version 1.1 for OS/2® Extended Edition (EASEL OS/2 EE) language.

IBM and OS/2 are registered trademarks of International Business Machines Corporation.

EASEL is a registered trademark of Interactive Images, Inc.

action Action Statement

Purpose: To reference an action routine.

Model

action ROUTINE_NAME

Where: ROUTINE_NAME is a string value representing the identifier for the action routine.

Remarks: The specified action routine must previously have been defined in the program. When the action routine is called, EASEL OS/2 EE performs the action statements specified in the routine, just as if the actions had been specified directly in the response definition.

All EASEL OS/2 EE variables, expressions, and built-in functions are evaluated when an action routine is executed, not when it is defined. Thus, an expression in the routine that has a certain value the first time the routine is called might have a different value the next time the routine is called. This would occur if the value of the variable used as an operand in the expression is changed between executions of the action routine.

You can reference an action routine inside the definition of another action routine. However, you must ensure that an action routine does not reference itself.

Examples:

```
response to start
    action Startup

response to KeyClass
    action parameter # The parameter contains the name
                    # of an action routine.
```

Errors: If the action routine references itself, an infinite loop may result.

See Also: **action is** Action Routine Definition

action bar Attribute Definition

Purpose: To define a region with an action bar.

Model

action bar AB_NAME

Where: AB_NAME is the identifier for a previously-defined action bar template.

Remarks: The action bar specification must appear within the object definition.

More than one region at a time can have action bars created from the same template.

Sense regions cannot have action bars.

Examples: action bar Part_Window_AB is
 pulldown View_Choices text "~View"
 choice View_Part text "~Part"
 choice View_Description text "~Description"
 choice View_Inventory text "~Inventory"
 end pulldown
end action bar

```
color 26 graphical region Part_Window
size 263 160
at position 60 80
in desktop
color 27 foreground
size border
title bar "Part"
system menu
horizontal scroll bar scroll by 6
vertical scroll bar scroll by 16
no scale
action bar Part_Window_AB
```

See Also: **action bar** is Template Definition

action bar is Template Definition

Purpose: To define an action bar template.

Model

action bar AB_NAME is

```
[PULLDOWN_DEF_OR_REF]...
[BUTTON_DEF_OR_REF]...
```

end action bar

Where: AB_NAME is the identifier for an action bar template.
 PULLDOWN_DEF_OR_REF is the definition of or reference to a pulldown.
 BUTTON_DEF_OR_REF is the definition of or reference to a button.

Remarks: The action bar template defines the structure and visual attributes of an action bar and its pulldowns. The template definition itself is not an on-screen object; the action bar only appears when it is specified as the action bar of a region.

The action bars in different regions can be created from the same template.

Use the **item** statement to reference pulldowns and buttons previously defined outside of an action bar template.

Buttons do not comply with CUA guidelines.

Examples:

```
action bar Part_Window_AB is
  pulldown View_Choices text "~View"
    choice View_Part text "~Part"
    choice View_Description text "~Description"
    choice View_Inventory text "~Inventory"
  end pulldown
end action bar
```

action bar is

color 26 graphical region Part_Window
size 263 160
at position 60 80
in desktop
color 27 foreground
size border
title bar "Part"
system menu
horizontal scroll bar scroll by 6
vertical scroll bar scroll by 16
no scale
action bar Part_Window_AB

See Also: **button** Definition
item Reference
pulldown Definition

action is Action Routine Definition

Purpose: To group a set of action statements into an action routine with its own identifier. This eliminates the repetition of any series of action statements.

Model

action ROUTINE_NAME **is**

[ACTION_STATEMENT]...

Where: ROUTINE_NAME is the identifier for the action routine.
ACTION_STATEMENT is an action statement.

Remarks: An action routine must be defined in every program in which it is to be used. The action routine can include any number of action statements and any type of action statements, even those which reference other action routines.
You can reference an action routine inside the definition of another action routine. However, you must ensure that an action routine does not reference itself.

Examples:

```
action Startup is
    enable Prompt
    make Screen_A visible
    enable Keys
    action StartFirst

action NextProgram is
    make Everything invisible
    make PleaseWait visible
    change to program Restart
```

See Also: **action** Action Statement

activate Action Statement

Purpose: To set the currently active region, dialog box, or dialog control.

Model

activate OBJECT_NAME

Where: OBJECT_NAME is the identifier for a framed region, dialog box, or dialog control.

Remarks: Activating a framed region places the region on top of its siblings with equal priority and changes the appearance of its title bar and size border. If the region has an active sibling, that sibling is deactivated. A region is automatically activated when the region is selected with the mouse.

Activating a dialog control moves the keyboard focus to that control. You can use this to direct the user to re-enter data into an entry field that has been entered incorrectly.

You can use this statement to activate modeless dialog boxes.

Examples: :
 default push button OK
 size 38 12
 at position 6 4
 in Logon_DialogBox
 group is Actions
 text "OK"

 entry field User_ID_Field
 size 52 14
 at position 96 63
 in Logon_DialogBox
 text size 10 columns
 left align

```
response to OK in Logon_DialogBox
  if (text of User_ID_Field = "") then
    activate User_ID_Field
  :
end if
```

add

add Action Statement

Purpose: To create a new object, a new action bar item, or a new action bar template during runtime.

Model #1 (for Objects and Items):

```
add { OBJECT_DEF | ITEM_DEF }
```

Model #2 (for Action Bar Templates):

```
add action bar AB_NAME is
```

```
    [PULLDOWN_DEF_OR_REF]...
```

```
    [BUTTON_DEF_OR_REF]...
```

```
end action bar
```

Where: OBJECT_DEF is the definition for an object.
ITEM_DEF is the definition for a pulldown, button, choice, or separator.
AB_NAME is the identifier for an action bar template.
PULLDOWN_DEF_OR_REF is the definition for, or reference to, a pulldown.
BUTTON_DEF_OR_REF is the definition for, or reference to, a button.

Remarks: The new object is created when the **add** statement is executed.

When using this statement, be careful to ensure that the identifier specified is unique. To minimize the risk of duplicating identifiers, specify a variable whose value is the identifier, rather than specifying an explicit name, in the **add** statement.

You can use an **item** reference to reference any item that you have defined outside of an action bar template.

Object definitions added at runtime must be added to classes in order to write responses for them.

Example #1: response to GoBack
 add key NewChoice at position 10 10
 solid red box 100 20
 move to 50 10
 text "ABANDON SCREEN"

 response to OpenWriter
 add textual region Blackboard
 window size 10 columns 10 lines
 at position 10 200 in GraphArea

Example #2: response to NewMessage
 add action bar Mail_Window_AB is
 pulldown Exit_Choices text "E~xit"
 choice Exit text "~Exit"
 choice Resume text "~Resume Mail"
 end pulldown
 end action bar

 add color 26 graphical region Mail_Window
 size 263 160
 at position 60 80
 in desktop
 color 27 foreground
 size border
 title bar "Mail"
 system menu
 horizontal scroll bar scroll by 6
 vertical scroll bar scroll by 16
 no scale
 action bar Mail_Window_AB

<i>See Also:</i> action bar Attribute Definition button Definition check box Definition choice Definition combination box Definition dialog box Definition dialog region Definition dropdown list Definition entry field Definition graphical region Definition group box Definition	image region Definition item Reference key Definition list box Definition multiline entry field Definition pulldown Definition push button Definition radio button Definition sense region Definition separator Definition static text Definition textual region Definition
---	---

add item to Action Statement

Purpose: To add an item to an action bar or pulldown at runtime.

Model

```
add item ITEM_NAME1 to
{ action bar | pulldown PD_NAME [from AB_NAME] }
in { REGION_NAME | REG_CLASS_NAME }
[ { before | after } item ITEM_NAME2 [from AB_NAME] ]
```

Where: ITEM_NAME1 is the identifier for a pulldown or button to be added to an action bar, or a choice or separator to be added to a pulldown.

PD_NAME is the identifier for a pulldown.

AB_NAME is the identifier for an action bar.

REGION_NAME is the identifier for a region.

REG_CLASS_NAME is the identifier for a class of regions.

ITEM_NAME2 is the identifier for an item in an action bar or pulldown.

Remarks: When adding to an action bar, if you don't specify the **before/after** clause, the item will be added to the end of the action bar. If you have buttons in an action bar, add any pulldowns before the first button item.

When adding to a pulldown, if you don't specify the **before/after** clause, the item will be added to the end of the pulldown.

Item definitions added at runtime must be added to item classes in order to write responses for them.

Examples: string Del_Char

```
action bar Phone_List_AB is
  pulldown File_Choices text "~File"
    choice Open text "~Open..."
    choice Save_As text "~Save As"
  end pulldown
end action bar
```

```
choice Delete text "~Delete"
integer LineNo

response to Phone_List_Box
  copy selected line from Phone_List_Box to LineNo
  extract from textual line LineNo from Phone_List_Box
  take 1 Del_Char
  if ( Del_Char = "*" ) then
    add item Delete to
      pulldown File_Choices from Phone_List_AB
      in Phone_Region
  end if
```

See Also: **action bar** Attribute Definition
 pulldown Definition

add to Action Statement

Purpose: To add drawing statements to the contents of an object.

Model #1 (for Graphical Objects):

```
add to { G_OBJECT_NAME | G_CLASS_NAME }  
  [at [position] X Y]  
  
    DRAWING_STATEMENT  
    [DRAWING_STATEMENT]...
```

Model #2 (for Textual Regions):

```
add to { TR_OBJECT_NAME | TR_CLASS_NAME }  
  [at [position] column COL_NO line LINE_NO]  
  
    DRAWING_STATEMENT  
    [DRAWING_STATEMENT]...
```

Model #3 (for List Boxes):

```
add to { LB_OBJECT_NAME | LB_CLASS_NAME }  
  
    INSERT_STATEMENT  
    [INSERT_STATEMENT]...
```

Where: G_OBJECT_NAME is the identifier of a graphical object.
G_CLASS_NAME is the identifier for a class of graphical objects.
X and Y represent the position in a graphical object at which these drawing statements begin.
DRAWING_STATEMENT is a drawing statement.
TR_OBJECT_NAME is the identifier for a textual region.
TR_CLASS_NAME is the identifier for a class of textual regions.

COL_NO and LINE_NO represent the column and line number (respectively) in the textual region at which the new drawing statements begin.

LB_OBJECT_NAME is the identifier for a list box object.

LB_CLASS_NAME is the identifier for a class of list box objects.

INSERT_STATEMENT is an insert string or insert file drawing statement.

<i>Use of Keywords:</i>	... at [position] ... For graphical objects, specifies the position in X Y coordinates. The graphics and text cursors move to the specified coordinates, and the drawing statements are added at that position. For textual regions, specifies the position in columns and lines. The text cursor moves to the specified column and line, and the drawing statements are added at that position.
<i>Remarks:</i>	The added contents are specified by the drawing statements or insert statements within this action statement. When specifying an add to statement for a class of objects, the class must be made up of all the same types of objects (graphical objects, textual regions, or list box objects).
<i>Default Values:</i>	If you do not specify a position for graphical objects or textual regions, the drawing statements are added at the current position of the graphics cursor or text cursor, respectively. For graphical regions, the current position of the graphics cursor is used if adding graphics, and the current position of the text cursor is used if adding text. Entries for list boxes are inserted according to the sorting order specified. If no sorting was specified, then entries are inserted at the end of the list box. If an insert file is done, then each line from the file is inserted according to the sorting order specified. If no sorting was specified, then the lines from the file are inserted at the end of the list box.
<i>Examples:</i>	response to Draw add to Plot draw to NewX NewY

add to

```
response to UpDate
  add to Heading at position 300 100
    " Today is: " date

response to StartReport
  add to Report
    at column 10 line 4
    insert "memorandum"
  :
response to OK
  add to Phone_List
    insert "555-1212"
```

See Also: **change** Action Statement

add to class Action Statement

Purpose: To add an existing object, or a class of objects, to a class of objects.

Model

add { OBJECT_NAME | CLASS_NAME } **to class** CLASS_NAME

Where: OBJECT_NAME is the identifier for an object.
CLASS_NAME is the identifier for a class containing objects.

Remarks: The objects in CLASS_NAME are added, and not the class itself.

The object, or class of objects, that is being added must already exist; it may have been previously defined, or it may have been previously created during runtime using the **add** action statement.

An object can be a member of many classes simultaneously. Adding an object to one class has no effect on the object's membership in other classes.

Examples: response to AnimalChoice
 add Monkey in Animals to class JungleBeasts

response to Small
 add Small to class Size

See Also: **add** Action Statement
class Definition

add to item class Action Statement

Purpose: To add an existing item, or a class of items, to a class of items.

Model

```
add item { ITEM_NAME [from AB_NAME] | ITEM_CLASS_NAME } to
item class ITEM_CLASS_NAME
```

Where: ITEM_NAME is the identifier for a pulldown, button, choice, or separator.

AB_NAME is the identifier for an action bar template. The action bar name must be specified if more than one template contains an item of the given name.

ITEM_CLASS_NAME is the identifier for a class containing items.

Remarks: The items in ITEM_CLASS_NAME are added, and not the class itself.

You can add standalone items or template items to an item class. You cannot add items from a particular region.

The item, or class of items, that is being added must already exist; it may have been previously defined, or it may have been previously created during runtime using the **add** action statement. An item can be a member of many classes simultaneously. Adding an item to one class has no effect on the item's membership in other classes.

Examples:

```
:
item class File_Options

response to Reset
  add item Open from Main_Window_AB to
    item class File_Options
:
```

See Also: **add** Action Statement
item class Definition

ancestry Response Inquiry Built-in Function

Purpose: To find and use the identifiers of the parent(s) of the object or item that stimulated the current response.

Model

ancestry

Remarks: For objects, EASEL OS/2 EE returns the fully-qualified name string of the parent(s) of the object that stimulated the response.

For items, EASEL OS/2 EE returns a string value representing the full ancestry specification of the object the item is in, including the object name.

The contents of **ancestry** are valid only until the response completes.

Examples: response to Done
delete ancestry

See Also: **ancestry of** Object Inquiry
Built-in Function
in Keyword
position Keyword

ancestry of Object Inquiry Built-in Function

Purpose: To find and use the identifiers of the parent(s) of a specified object.

Model

ancestry of OBJECT_NAME

Where: OBJECT_NAME is a string value representing the identifier of an object.

Remarks: The value returned is the fully-qualified name string of the parent(s) of the specified object.

The syntax used in the returned value for the ancestry is as follows, with child defined as:

CHILD in PARENT_N in ... in PARENT2 in PARENT1

When **ancestry of** CHILD is specified, EASEL OS/2 EE returns:

PARENT1/PARENT2/.../PARENT_N

The string returned indicates that PARENT_N is in the object PARENT2, which is in PARENT1, which is in the coordinate system of the screen as defined by the **screen size** Environment Declaration .

Examples: response to AddWindow
 add blue graphical region Writer
 size 510 280
 at position XCalc YCalc
 in ancestry of BigWindow

See Also: **ancestry** Response
 Inquiry Built-in Function
 in Keyword
 position Keyword

append Action Statement

Purpose: To append one or more string values to a string variable.

Model

append VALUE1 [VALUE2]... **to** STRING_VAR_NAME

Where: VALUE1 and VALUE2 are any string values.
STRING_VAR_NAME is the identifier for a string variable.

Remarks: The new value or values are appended to any existing contents of the specified string variable.
You cannot append values to an array element.
You can append text lines, a text segment, or a text block from a textual region to a variable.
If you specify a value of a different type, EASEL OS/2 EE will try to convert the value to the expected type. See "Type Conversions" on page 1-348.

Examples: response to Plus5
 append ".05" to Dollars

 response to line "Full" from Host
 append "Full" to Status

 action MoreLocs is
 append MA FL to Citylist

See Also: **length of** Special Inquiry Built-in Function
variable Definition

application Environment Declaration

Purpose: To specify a local or remote application that will be used in the EASEL OS/2 EE program.

Model

application APP_NAME

Where: APP_NAME is an identifier for the name of the application program.

Remarks: You must define all application programs with which your EASEL OS/2 EE program will communicate, before they can be recognized by EASEL OS/2 EE. The form of the **application** statement is the same whether the application is local or remote.

The name that you specify in the **application** statement is a logical name and is not necessarily the name by which the operating system recognizes the program. You can specify any name that you wish.

Once an application program has been defined, you refer to it throughout your EASEL OS/2 EE program by specifying the name provided in the **application** statement.

Examples: application Database

application S

See Also: **start** Action Statement
stop Action Statement

array Definition and Reference

Purpose: An array is a group of variables or constants, all accessed by the same name but with different subscripts. An array element is used like a regular string, integer, or boolean variable or constant, except that an array element cannot be appended to.

Model

```
[global] { integer | string | float | boolean }
[ variable | constant ]
{ ARR_NAME[ N ] |
  ARR_NAME[ N1[, N2]... ] |
  ARR_NAME[ NA thru NB ] |
  ARR_NAME[ N1A thru N1B[, N2A thru N2B]... ] } [is
  VAR1[, VAR2]... ]
```

Where: ARR_NAME is the identifier for the variable or constant array.

N is an integer value representing the number of elements in the array, starting with 1 and ending with N.

N1[, N2]... are integer values representing the number of elements for each dimension of the array, starting with 1 and ending with N1[, N2]... respectively. The number of dimensions must not exceed 11.

NA thru NB are integer values representing the beginning and ending integers used to reference the elements of the array.

N1A thru N1B[, N2A thru N2B]... are integer values representing the beginning and ending integers used to reference the elements for each dimension of the array. The number of dimensions must not exceed 11.

VAR1[, VAR2]... are integer, floating point, string, or boolean values representing the initial values of the array elements.

Note: Beginning and ending brackets shown with ARR_NAME in the model ([]) must be specified exactly as shown.

array

Use of

Keywords:

global ...

Defines a global variable array. When global arrays are loaded from global space into program space for an EASEL OS/2 EE program that is executed by a **change to program** statement, the new number of dimensions is *not* checked against the number of dimensions defined for the array in the original program. Therefore, if the number of elements for any dimension or the number of dimensions of the array in the new program is different than was defined in the original program, the results are unpredictable.

... is VAR1[, VAR2] ...

Specifies initial values for the array elements. If too few initial values are given, the defaults shown below are used to fill the remainder of the array. If too many initial values are given, the extra values are dropped. In either case, a warning is generated. When EASEL OS/2 EE initializes an array, it cycles through all the subscripts in the array reference, varying the last dimension's subscript first.

Remarks:

When defining an array, you must specify the data type, the array name, and the range of permissible elements (subscripts) of the array. The subscripts for arrays are always surrounded by brackets. The maximum number of elements per array depends on the type of array, as follows:

- 16K integers or strings
- 8K floats
- 512K booleans

An array can have up to eleven dimensions. Only the first dimension can be resized, using the **resize** action statement.

*Referencing
an Array:*

When referencing an array, the [N] becomes the subscript that references an array element, and does not designate the range of permissible elements (subscripts) as it did when you defined it.

When referencing a multiple dimension array, the [N1[, N2]...] becomes the subscripts for each dimension of the array that references a single array element.

*Default
Values:*

Element	Default Initializations
boolean	false
integer	0
float	0.00
string	""

Examples: global integer Profits[5]

 global string Sales [2,3] is # Sales[1,2] = "DC"
 "CT","DC","NY","BC","SF","LA" # Sales[2,1] = "BC"

 integer Bar[0 thru 2,10 thru 12] is
 2,3,3,Var_A,15,(5*Var_A),5,3,1228

 # The following two examples use array references:

 copy Apples[2,2] to Accumulate

 copy Sales[Person,Region] to City

See Also: **resize** Action Statement

background at Object Inquiry Built-in Function

See: **foreground at** and **background at** Object Inquiry Built-in Functions on page 1-152.

background of Object Inquiry Built-in Function

Purpose: To find and use the background color of a specified graphical, image, or textual region.

Model

background of REG_NAME

Where: REG_NAME is a string value representing the identifier for a graphical, image, or textual region.

Remarks: The **background of** function can be applied to graphical, image, and textual regions only. The value returned is the integer value associated with the background color of the specified region.

Examples: response to Mute
make Labels color background of Bulletin

See Also: **color** Keyword
foreground at and **background at** Object Inquiry Built-in Functions
foreground of Object Inquiry Built-in Function

begin BLOCK Action Statement

Purpose: To group a number of response definitions within one action statement. Using blocks, you can indicate when responses are appropriate and when they are inappropriate.

Model

```
begin [ guarded | resumable ] [BLOCK_NAME]
```

```
    [RESPONSE_DEFINITION]...
```

```
end [BLOCK_NAME]
```

Where: BLOCK_NAME is the identifier for the block.

RESPONSE_DEFINITION is a response definition.

Use of **begin guarded ...**
Keywords: EASEL OS/2 EE ignores all stimuli it receives outside of the specified block while the block is being executed. EASEL OS/2 EE holds keyboard and application input that is not applicable to the responses within the block, making them available when the guarded block exits. Selections will not be taken for objects that do not have responses within that block, no matter what level of nesting they occupy.

A guarded block can be left only by the explicit statements **leave block**, **leave loop** (if the guarded block is within a loop), **exit** (which must be specified within the block), or **response to low memory**.

A guarded block cannot contain a **response to low memory** or a **response to termination** response definition. Specifying a **response to termination** in a guarded block does not result in an error message, but will have no effect.

begin resumable ...

EASEL OS/2 EE leaves the block in response to any stimulus outside the block and then returns to the block immediately after that response has been executed. EASEL OS/2 EE will first finish any current action statements in the block, then execute the actions in the response that caused the block to be left, and lastly return to the block and await a stimulus from within that block.

Resumable blocks cannot contain **response to low memory** or **response to termination** response definitions.

Remarks: A **begin** BLOCK is an action statement that contains a group of response definitions. You can define and use a block anywhere an action statement can be specified, including within action routines, loops, and conditional (if/else) action statements. The **begin** statement begins a block. The **end** statement ends a block. The response definitions must lie between the **begin** and **end** of the block.

If you use an identifier in the **begin** statement, there must be nothing else in the program with that name. You must also specify the identifier name in the **end** statement, exactly as in the **begin** statement. You can completely omit the identifier from the definition.

A response definition that is inside a block might itself contain another block as part of its set of action statements; these are called *nested* blocks. Nested blocks create a program that contains many levels of responses, ranging from the innermost block to the outermost.

You cannot specify action statements directly in blocks. All action statements must be specified in response definitions.

Examples:

```

response to char "start" from Remote
begin
:
end

response to Plot
begin FirstChoice
:
end FirstChoice

```

begin

```
response to PlotTwice
  begin resumable Always
  :
  end Always

response to line from Remote
  begin guarded
  :
  end
```

See Also: **leave block** Action Statement
 response to OBJECT Response Definition

blank block Drawing Statement

Purpose: To overwrite a rectangular area of text in a textual region with blanks.

Model

```
blank block [column] COL_NO [line] LINE_NO  
[thru [column] COL_NO [line] LINE_NO]
```

Where: COL_NO are integer values representing the starting and ending column numbers in the textual region.

LINE_NO are integer values representing the starting and ending line numbers in the textual region.

Remarks: Blanks are displayed in the background color of the textual region.

Examples: response to More
 add to FormerText
 blank block column 3 line 1
 thru column 8 line 3

See Also: **block** Definition (for Text)

block Definition (for Text)

Purpose: To specify a block of text in a textual region.

Model

block [**column**] COL_NO [**line**] LINE_NO
[**thru** [**column**] COL_NO [**line**] LINE_NO]

Where: COL_NO are integer values representing the starting and ending column numbers in the textual region.

LINE_NO are integer values representing the starting and ending line numbers in the textual region.

Remarks: This syntax specification can be used only with textual regions, with the **insert**, **overwrite**, and **blank** drawing statements, the **make block** and **remove** action statements, and the **textual** built-in function.

A block is any rectangular area within a textual region. It is defined by any two opposite corner positions, as follows, and includes the specified corner positions:

upper-left followed by lower-right
lower-right followed by upper-left
upper-right followed by lower-left
lower-left followed by upper-right

A block can include character positions which do not contain any actual text.

Color attributes are preserved in blocks from colored textual regions. (If the block is colored but the destination textual region is not, color attributes are dropped.)

If the keywords **column** and **line** are not included, the first and third integer values are used as column values, and the second and fourth integer values are used as line values.

If you provide only one set of column and line numbers, the block will be a single character position.

Examples: block column 8 line 3 thru column 3 line 1 from Writer

block 3 1 thru 8 3 from Writer

Errors: If a **block** is specified for a class that has objects other than textual regions, or for any object other than a textual region, an error message will be generated.

See Also:

blank block Drawing Statement	overwrite Drawing Statement
insert Drawing Statement	remove Action Statement
make block and make segment COLOR Action Statements	segment Definition
	textual Object Inquiry Built-in Function

border Attribute Definition

Purpose: To define a border for a graphical region, image region, textual region, dialog region, **shape** drawing statement, or dialog box.

Model #1 (for Graphical, Image, and Textual Regions):

[**size** | [**color**] COLOR] **border**

Model #2 (for Dialog Regions):

[**size**] **border**

Model #3 (for Shapes):

[[**color**] COLOR] **border**

Model #4 (for Dialog Boxes):

[**dialog**] **border**

Where: COLOR is a string or integer value representing a color.

Use Of **size** ...

Keywords: Specifies that the graphical, image, textual, or dialog region will be surrounded by the standard Presentation Manager™ (PM) sizing border that can be used to resize the region.

[**color**] COLOR ...

Specifies that the graphical region, image region, textual region, or shape will be surrounded by a one-pixel border of the color indicated.

Presentation Manager is a trademark of International Business Machines Corporation.

dialog ...

Specifies that the dialog box will be surrounded by the standard PM dialog border.

Remarks: The **border** keyword must appear within the object definition or **shape** drawing statement.

If other PM frame attributes are specified for the graphical, image, or textual region, then the **border** specification will cause a one-pixel wide border in the PM border color to be displayed.

A border color cannot be specified for a graphical, image, or textual region that has an ancestry of **in desktop**. If the region also does not have any other frame attributes, **border** cannot be specified either.

Default Values: *Graphical, Image, and Textual Regions:* The default border color is white. If no border is specified, the default is an invisible white border.

Dialog Regions: If **border** is specified, the standard PM window border will be drawn.

Shapes: The default border color is white. If no border is specified, the shape will be filled with the specified color, and no border will be drawn.

Dialog Boxes: If **border** is specified, the standard PM window border will be drawn.

Examples: blue graphical region Draw
size 100 100 at position 30 30
red border

```
key Go at position 10 10
solid blue shape
red border
boundary ...
end shape
```

```
dialog box Phone_DB
size 200 110
at position 50 69
dialog border
title bar "Dialog Box"
system menu
```

See Also: **border** of Object Inquiry
Built-in Function
color Keyword
dialog box Definition
dialog region Definition

graphical region Definition
image region Definition
textual region Definition
shape Drawing Statement

border of Object Inquiry Built-in Function

Purpose: To find and use the color of the border of a specified graphical region, image region, or textual region.

Model

border of REG_NAME

Where: REG_NAME is a string value representing the identifier for a region.

Remarks: The **border of** function can be applied to a graphical, image, or textual region. The value returned is the integer value of the color associated with the border of the specified region.

If the border is invisible, EASEL OS/2 EE returns a negative integer value of the border color.

Default Values: If no border or a size border was defined for the object, EASEL OS/2 EE returns the value **-7** (for **invisible white** border, the default).

Examples: response to NewScreen
 add graphical region Go2
 size 20 20 at position 10 10
 color (border of Go) border

See Also: **border** Attribute Definition
color Keyword
make border COLOR Action Statement

bottom [of] Object Inquiry Built-in Function

See: **top [of]** and **bottom [of]** Object Inquiry Built-in Functions on page 1-345.

box Drawing Statement

Purpose: To draw a rectangle in a graphical object.

Model

[solid] [[color] COLOR] box XOFFSET YOFFSET

Where: COLOR is a string or integer value representing a color.

XOFFSET is an integer value specifying the width of the box, in the object's coordinates.

YOFFSET is an integer value specifying the height of the box, in the object's coordinates.

Use of **solid ...**
Keywords: Fills the box with the box's color.

[color] COLOR ...
Draws the box in the specified color, overriding all other color specifications that otherwise would affect this drawing statement.

Remarks: EASEL OS/2 EE positions the lower left corner of the box at the current position of the graphics cursor. The position of the opposite (upper right) corner is determined by the XOFFSET and YOFFSET specifications. After the box is drawn, the graphics cursor has returned to its initial position, and the text cursor is moved to this position.

The outline of a box is always one pixel wide.

Default If **solid** is not specified, just the box's perimeter (outline)
Values: will be drawn.

If color is not specified, the box will be drawn in the color specified in a pattern reference or object definition.

Examples: key Start at position 10 10
 box 50 25

key Start at position 100 100
 solid red box Length Height

response to ToDo
 add to Start
 solid color Color box
 xposition of Stop yposition of Stop

Built-in Functions

Purpose: EASEL OS/2 EE's predefined built-in functions allow you to inquire about the attributes of an object or item, or the nature of a stimulus during execution. As the answer to your inquiry, EASEL OS/2 EE returns a value for the function, based upon the position or contents of the object or item, or the nature of the stimulus.

Remarks: All built-in function names are keywords and therefore begin with lowercase letters. Built-in functions can be used anywhere a value of the appropriate data type is expected.

Response Inquiry functions allow you to inquire about the characteristics of the stimulus that initiated a response.

Object Inquiry functions allow you to inquire about the characteristics of a specified object. Object inquiry functions that require internal calculations (**xsize**, **ysize**, **column size**, and **line size**) will not return the correct value when used in the definition of a compile-time object. (These calculations are not made until runtime.)

For those object inquiry functions that are followed by the optional keyword **of**, you can omit the object name when the function is specified in a drawing statement. In this case, the object defaults to the current object. If any of these functions is specified *without* an object name in any statement *other* than a drawing statement, an error message will be generated. Always specify the object name for the function, even if the syntax allows you to omit it.

Item Inquiry functions allow you to inquire about the characteristics of a specified item.

Action Inquiry functions allow you to determine the success or failure of certain statements related to textual regions.

Special Inquiry functions allow you to inquire about various items such as the current time or date, or the number of characters in a string variable.

Following is a summary of all the built-in functions.

Response Inquiry Built-in Functions

Function	Value Returned
ancestry	<i>Objects:</i> String containing the ancestry of the selected object. <i>Items:</i> String containing the object name that contains the item, with its full ancestry.
eventnumber	Integer representing the event number provided by the Dynamic Link Library (DLL) invoking the response.
eventparam	Integer representing the event parameter provided by the DLL invoking the response.
input	String received from an application or from the keyboard.
object	String containing the name of the object or item for which a response is taken.
parameter	String containing the selected object's or item's parameter.
xcoord, ycoord	<i>Textual Regions:</i> Column and line number of the character selected. <i>Graphical Objects:</i> X and Y coordinates of the point selected, in the coordinate system of the object for which a response is taken.

Object Inquiry Built-in Functions

Function	Value Returned
ancestry of	String containing the object's ancestry.
background at	<i>Textual Regions:</i> Integer that is the background color of the specified line and column of text.
background of	Integer that is the color of a region's background. Not applicable to sense regions.
border of	Integer associated with the color of a region's border. Not applicable to sense regions or dialog regions.
checked in group	<i>Radio Buttons:</i> String containing the object name of the radio button currently checked in the specified group.
column size of	<i>Textual Regions:</i> Width of a column, in the parent's coordinates.

Function	Value Returned
exists	True if the object exists; false otherwise.
foreground at	<i>Textual Regions:</i> Integer that is the foreground color of the specified line and column of text.
foreground of	Integer that is the color of the key, or the color of the region's foreground. Not applicable to sense regions.
handle of	Integer that is the PM handle for the object. Not applicable to keys or sense regions.
is checked	<i>Check Boxes and Radio Buttons:</i> True if the object is checked; false otherwise.
left [of], right [of]	<i>Textual Regions:</i> Column numbers of the first column (always 1) and the last column of the longest line in the textual region. <i>Entry Fields:</i> left is always 1; right is the length of the text in the entry field. <i>Graphical Objects:</i> X coordinates of the leftmost and rightmost points occupied by the contents of the object.
line size of	<i>Textual Regions:</i> Height of line, in the parent's coordinates.
parameter of	String containing the object's parameter.
priority of	Integer value that is the priority of the object.
selectability of	True if the object is selectable; false otherwise.
selected line from	<i>List Boxes:</i> Integer value that is the selected line from a single-select list box.
text of	<i>Regions:</i> String containing the title bar text. <i>Dialog Controls:</i> String containing the text displayed in the control.
textual { line LINE_NO 	SEGMENT_DEF BLOCK_DEF } from TR_NAME
	<i>Textual Regions:</i> String containing the contents of the specified line, segment, or block.
textual line from	<i>List Boxes:</i> String containing the contents of the specified line.

Function	Value Returned
top [of], bottom [of]	<i>Textual Regions and List Boxes:</i> Line numbers of the first line (always 1) and the last line of the textual region. <i>Graphical Objects:</i> Y coordinates of the topmost and bottommost points occupied by the contents of the object.
visibility of	True if the object is visible; false otherwise.
window xposition [of], window yposition [of]	<i>Textual Regions:</i> Column number of the leftmost character in the window, and line number of the topmost character in the window. xposition is column number; yposition is line number. <i>Graphical, Image, Sense, and Dialog Regions:</i> X and Y coordinates of the starting position of the region's window (expressed in dialog units for dialog regions).
window xsize [of], window ysize [of]	<i>Textual Regions:</i> Height of the window in lines, and width of the window in columns. xsize is width; ysize is height. <i>Graphical, Image, Sense, and Dialog Regions:</i> X and Y dimensions of the region's window size (expressed in dialog units for dialog regions).
xcursor [of], ycursor [of]	<i>Textual Regions:</i> Column and line number of the position of the text cursor for the object. xcursor is column number; ycursor is line number. <i>Graphical Objects:</i> X and Y coordinates of the position of the graphics cursor for the object.
xmiddle [of], ymiddle [of]	<i>Textual Regions:</i> xmiddle is the last column of the longest line in the textual region divided by 2; ymiddle is the number of lines in the textual region divided by 2. <i>Graphical Objects:</i> xmiddle is the sum of the leftmost point plus the rightmost point, divided by 2; ymiddle is the sum of the topmost point plus the bottommost point, divided by 2.
xposition [of], yposition [of]	X and Y coordinates of the lower left corner of the viewport, in the parent's coordinates.
xsize [of], ysize [of]	X and Y dimensions of the region's viewport, in the parent's coordinates.

Item Inquiry Built-in Functions

Function	Value Returned
is checked	True if the item is checked; false otherwise.
is enabled	True if the item is enabled; false otherwise.
text of item	String containing the text displayed in the specified item.

Action Inquiry Built-in Functions

Function	Value Returned
errorlevel	Code returned by an external subroutine, start statement, or send statement; 0 if no error, otherwise an error code is returned.
found	<i>Textual Regions:</i> True if the result of a find statement was successful; false otherwise.
ioerror	<i>Image Regions, Textual Regions, List Boxes:</i> True if the result of a read or write action statement, or an insert file drawing statement was unsuccessful; false otherwise.

Special Inquiry Built-in Functions

Function	Value Returned
clipboard	String containing the text from the clipboard.
date	String containing the current date, in MM/DD/YY format.
freesize	Integer that is the byte size of the single largest available block of memory that EASEL OS/2 EE can use.
length of	Integer that is the number of characters in the string variable.
marker	Integer that is the column number of the pointer's position in a string. Must be used with extract from action statement.
member	String containing the name of the current member of the class in a for each member loop .
members of	String containing the names of all members in the class.

Built-in Functions

Function	Value Returned
selected line	Line number of the current selected line in a for each selected line loop from a multiple selection list box.
time	String containing the current time, in HH:MM:SS format.

button Definition

Purpose: To define an action bar button item.

Model

```
[ enabled | disabled ] button BUTTON_NAME text BUTTON_TEXT
  [accelerator is [ { control | alt | shift }
                  [ control | alt | shift ]... ] ACCEL_KEY]
  [parameter is PAR_STRING]
```

Where: BUTTON_NAME is the identifier for the button.

BUTTON_TEXT is a string value representing the text that will appear in the button. This text can be any string. The text can contain the character ~ to indicate the mnemonic character.

ACCEL_KEY is an integer value selected from a predefined list found in an include file (**accel.inc**), representing the associated accelerator key. The **accel.inc** file assigns integer constant names to each of the integer values, and these constant names can be used if this file is included. You can combine the accelerator key with any number of the status flags **control**, **alt**, or **shift**.

PAR_STRING is a string value representing the parameter of the button.

*Use of
Keywords:*

text ...

Specifies the text associated with the button. This text will be displayed in the action bar. If a character in the string is preceded by the ~ character, that character is used as the mnemonic for the button and will be underlined in the text of the button.

accelerator is ...

Specifies the accelerator key that the user can press to select the button.

... control ...

Specifies that the user must hold down the Control key when pressing the key defined in ACCEL_KEY.

... alt ...

Specifies that the user must hold down the Alt key when pressing the key defined in ACCEL_KEY.

button

... **shift** ...

Specifies that the user must hold down the Shift key when pressing the key defined in ACCEL_KEY. The ACCEL_KEY character string is case-independent. It is the **shift** keyword that determines that the accelerator key will be uppercase.

Remarks: Typically, a button is used to display the "F1 = Help" message in the action bar of the primary window.

Buttons do not comply with CUA guidelines.

To use the constant names in the **accel.inc** file, be sure to include the file. (See "accel.inc File" on page 4-3 for a list of the valid names.)

The parameter of an item is attached to, and is an attribute of, the item defined in the action bar template or the standalone item template. If the parameter is changed in an action statement, it is changed in the template definition and everywhere it is referenced.

Button items can be defined within an action bar or as a standalone template. As a standalone template, it can be referenced (and therefore included) in any action bar template.

Example: include accel.inc

```
action bar View_Mail_AB is
  pulldown Exit text "~Exit"
    choice Exit_Choice text "E~xit"
    choice Resume_Choice text "~Resume"
  end pulldown
  enabled button Help text "F1 = Help"
    accelerator is KEY_F1
end action bar
```

See Also: **action bar** is Template Definition
accel.inc File (Chapter 4)
item Reference

call Action Statement

Purpose: To call a subroutine.

Model

call SUBROUTINE_NAME([ARG1[, ARG2]...])

Where: SUBROUTINE_NAME is the identifier for a previously defined subroutine.

ARG1[, ARG2]... are the argument variables of the subroutine. (They must be enclosed in parentheses.)

Remarks: The **call** statement syntax is the same for EASEL OS/2 EE subroutines and for library subroutines.

The argument variables used in the **call** statement (ARG1[, ARG2]...) must be the same number and type as those specified in the subroutine declaration or definition.

Examples: response to item Open in Main_Window_AB
 copy true to New_Mail
 copy "" to Message_Name
 call SetupMail_DB(New_Mail, Message_Name)

See Also: **subroutine** Declaration

change Action Statement

Purpose: To replace all the contents (drawing statements) of an object, or the contents of all the objects in a class.

Model

change [**graphics**] { OBJECT_NAME | CLASS_NAME } **to**
[DRAWING STATEMENT]...

Where: OBJECT_NAME is the identifier for an object.
CLASS_NAME is the identifier for a class of objects.
DRAWING_STATEMENT is a drawing statement.

Use of **change graphics ...**
Keywords: Replaces the contents of an object and adds the new drawing statements, but does not delete, or affect in any way, any of the object's children. Thus, you need not first delete the object's children from any classes to which they belong.

Remarks: When the keyword **graphics** is omitted, EASEL OS/2 EE deletes the existing drawing statements for the object, deletes all of the object's children, and adds the new drawing statements starting at the object's origin. This statement has the same effect as using the **clear** statement followed by the **add to** statement.

Since a textual region's drawing statements are distinct from graphical drawing statements, do not specify a **change** statement for a class of objects that includes textual regions as well as graphical objects.

Before you replace the contents of an object, it is recommended that you first delete its children from any classes to which they belong, using the **delete from class** action statement, unless the **graphics** keyword is used.

Examples: response to StopKey
 change StopKey to
 :
response to PanelOpts
 change graphics Start in Panel to
 :

See Also: **clear** Action Statement
 delete from class Action Statement

change action bar Action Statement

Purpose: To change which action bar is associated with a region or with all the regions in a class.

Model

change { REGION_NAME | REG_CLASS_NAME } action bar to AB_NAME

Where: REGION_NAME is the identifier for the region (graphical, image, or textual) to be changed.
REG_CLASS_NAME is the identifier for a class of regions.
AB_NAME is the identifier for an action bar template. The action bar of the specified region is generated from this template.

Examples: response to item View_Mail from Main_Window_AB
change Secondary_Window action bar to Mail_AB

See Also: **action bar is** Template Definition

change position Action Statement

Purpose: To change the position of an object.

Model

```
change { OBJECT_NAME | CLASS_NAME } position
  { by XOFFSET YOFFSET | to X Y [in PARENT_NAME] }
```

Where: OBJECT_NAME is the identifier for an object.

CLASS_NAME is the identifier for a class of objects.

XOFFSET and YOFFSET are integer values representing the number of relative coordinates by which the object is being moved, in the parent's coordinates.

X and Y are integer values representing the absolute coordinates of the object's new origin.

PARENT_NAME is the identifier for the object's new parent.

Use of Keywords:

- ... **by** XOFFSET YOFFSET
Moves the object's origin by relative X and Y coordinates from its current position, in the coordinate system of its parent.
- ... **to** X Y
Moves the object's origin to the absolute X and Y coordinates specified, in the coordinate system of its parent.
- ... **in** PARENT_NAME
Removes the current parent of the object, and places the object in the coordinate system of the specified parent. If no parent is specified, the object will remain in the coordinate system of its parent.

Examples:

```
response to Sel2
  change Square position by 50 100

response to Red
  change Square position to 110 120

response to Menu3
  change Button1 in Menu1 position by -50 0

response to Colder
  change Zipper position to 100 300 in NewCoat
```

change position

See Also: **ancestry** Response Inquiry Built-in Function
 ancestry of Object Inquiry Built-in Function
 change window position Action Statement
 position Keyword

change priority Action Statement

Purpose: To change the priority of an object.

Model

change { OBJECT_NAME | CLASS_NAME } **priority to** PRIORITY

Where: OBJECT_NAME is the identifier for an object.
CLASS_NAME is the identifier for a class of objects.
PRIORITY is an integer value representing the object's new priority.

Remarks: The priority of the primary region and regions whose parent is desktop cannot be changed.

Examples: response to Help
change Button1 priority to 3

response to HelpSet
change Button2 in Keyboard priority to New_priority

See Also: **priority is** Attribute Definition
priority of Object Inquiry Built-in Function

change size Action Statement

Purpose: To change the viewport size of a graphical, image, textual, or dialog region, dialog box, class of regions, or class of dialog boxes.

Model

change { REG_NAME | REG_CLASS_NAME } **size** { **by** | **to** } X Y

Where: REG_NAME is the identifier for a region or dialog box.

REG_CLASS_NAME is the identifier for a class of regions or dialog boxes.

X and Y are integer values representing the X and Y coordinates (respectively) by which, or to which, the viewport's size is to be changed. Both integer values are specified in the coordinates of the region's parent.

Use of **change size by ...**

Keywords: Changes the size by the specified coordinates relative to the current viewport size of the region.

change size to ...

Changes the viewport size to the absolute specified coordinates.

Remarks: Using the **change size** statement for a textual region, dialog region, dialog box, or a region with the **no scale** attribute specified also changes its window size.

Examples: response to ReducePad
change Pad size by 30 60

See Also: **change window size** Action Statement

change text Action Statement

Purpose: To change the displayed text of a dialog box, dialog control, region's title bar, or item.

Model #1 (for Dialog Boxes, Dialog Controls, and Region's Title Bars):

change OBJECT_NAME **text to** TEXT_STRING

Model #2 (for Items):

```
change item { ITEM_NAME [from AB_NAME] | ITEM_CLASS_NAME }
in { REGION_NAME | REG_CLASS_NAME }
text to ITEM_TEXT
```

Where: OBJECT_NAME is the identifier for a dialog box, dialog control containing text, or region containing a title bar.

TEXT_STRING is a string value representing the new text to be displayed in the object or title bar.

ITEM_NAME is the identifier for a pulldown, button, or choice.

AB_NAME is the identifier for an action bar template.

ITEM_CLASS_NAME is the identifier for an item class.

REGION_NAME is the identifier for a region containing the item.

REG_CLASS_NAME is the identifier for a class of regions.

ITEM_TEXT is a string value representing the new text that should appear in the item. This text can be any string.

Examples: response to Report_Names_TR
 copy textual line ycoord from Report_Names_TR to
 Window_Title
 change View_Window text to Window_Title

change text

See Also:

action bar Attribute
Definition
button Definition
check box Definition
choice Definition
combination box
Definition
dialog box Definition
dialog region Definition
dropdown list Definition
entry field Definition

graphical region Definition
group box Definition
image region Definition
multiline entry field
Definition
pulldown Definition
push button Definition
radio button Definition
separator Definition
static text Definition
textual region Definition

change to program Action Statement

Purpose: To transfer control from the current program to another EASEL OS/2 EE program.

Model

change to program PROGRAM_NAME

Where: PROGRAM_NAME is a string value representing the name of the EASEL OS/2 EE program to which control is to be transferred.

Remarks: When EASEL OS/2 EE executes a **change to program** statement, it causes the current program to terminate. The display is cleared, and all objects, variables, constants, classes, patterns, and responses belonging to the program that is being exited cease to exist. Note, however, that global variables still exist, started applications continue to run, and open selection devices remain open. The new program begins execution with its **response to start** statement.

In most cases, the **change to program** statement should be the last action statement in the response definition in which it is specified, since any following statements will not be executed unless an error condition exists.

If EASEL OS/2 EE cannot find the program specified in PROGRAM_NAME, it will execute any action statements that follow the **change to program** statement. This can be useful for error detection purposes.

The value for PROGRAM_NAME must be a string value that contains the name of the program (with or without the .EBI extension). It does not have to be in the same directory as the program that is being terminated; you can specify a full directory path name for it. When specifying a path, include an *extra* backslash each time one is specified (since one backslash is the escape character in strings).

change to program

Global Variables in Changed Programs:

Ordinarily, initialization of global variables is specified only in the first program. If any subsequent program initializes the global variable (for example, "global variable X is 100"), the initialization is *ignored*, except in the case of a new global variable. Therefore, do not initialize global variables in any program that will be run with the **change to program** statement. Even when control is transferred to a program that was created with the **save program as** statement, the **change to program** statement transfers the value of its current global variables to the new program.

When a **change to program** statement is executed, new global array dimensions are *not* checked against the old dimensions. Therefore, if the number of elements for any dimension or the number of dimensions of the array in the new program is different from what was defined in the original program, the results are unpredictable.

Example: response to Database_Key
 send ExitCode to CurrentApp
 change to program "Database"

 response to GoBackKey
 change to program "c:\\cuart-ee\\first"

See Also: **exit** Action Statement

change window position Action Statement

Purpose: To change the position of the window of a graphical region, image region, sense region, textual region, dialog region, or list box.

Model #1 (for Graphical, Image, Sense, and Dialog Regions):

```
change { OBJECT_NAME | CLASS_NAME } window position
{ [by] XOFFSET YOFFSET | to X Y }
```

Model #2 (for Textual Regions and List Boxes)

```
change { OBJECT_NAME | CLASS_NAME } window position
{ [by] COLUMNS [columns] LINES [lines] |
to [column] COL_NO [line] LINE_NO }
```

Where: OBJECT_NAME is the identifier for an object.

CLASS_NAME is the identifier for a class of objects.

XOFFSET and YOFFSET are integer values representing the coordinates by which the window's position is being changed.

X and Y are integer values representing the coordinates to which the window's position is being changed.

COLUMNS and LINES are integer values representing the number of columns and lines by which the window's position is being changed.

COL_NO and LINE_NO are integer values representing the column and line number to which the window's position is being changed.

Use of Keywords: **change window position [by] ...**
Changes the relative position of the window by the positive and/or negative values specified.

change window position to ...
Changes the absolute position of the window to the values specified. Values cannot be negative.

change window position

Remarks: When changing the window position of a list box, the column position is ignored. It will always be column 1 when changing **to**, and 0 columns when changing **by**.

Example:

```
response to NewPad
  change Row window position      # two graphical
  to 10 75                        # regions

response to Mover
  change Graph in Display window position # graphical
  by 50 100                          # region

response to Rewrite
  change Report window position      # textual region
  by 10 columns -5 lines
```

See Also: **change position** Action Statement

change window size Action Statement

Purpose: To change the size of the window of a graphical, image, sense, or textual region.

Model #1 (for Graphical, Image, and Sense Regions):

```
change { REGION_NAME | REG_CLASS_NAME } window size
  [ by | to ] X Y
```

Model #2 (for Textual Regions):

```
change { TR_NAME | TR_CLASS_NAME } window size
  [ by | to ] COLUMNS columns LINES lines
```

Where: REGION_NAME is the identifier for a region.

REG_CLASS_NAME is the identifier for a class of regions.

X and Y are integer values representing the X and Y coordinates (respectively) by which, or to which, the graphical, image, or sense region's window size is to be changed.

COLUMNS and LINES are integer values representing the number of columns and lines by which, or to which, the textual region's window size is to be changed.

TR_NAME is the identifier for a textual region.

TR_CLASS_NAME is the identifier for a class of textual regions.

Use of
Keywords:

change window size by ...

Changes the relative size of the window by the values specified. For graphical, image, or sense regions, integer values represent the number of X Y coordinates by which the window size will be changed; for textual regions, integer values represent the number of columns and lines by which the window size will be changed. If negative values are used for regions, the result must be positive.

change window size

change window size to ...

Changes the absolute size of the window. Values are positive integer values representing the number of X Y coordinates of the new window size (for graphical, image, or sense regions), or the number of columns and lines of the new window size (for textual regions).

Remarks: The window lies in the coordinate system of the object.
Using the **change window size** statement for a textual region also changes its viewport size.

The **change window size** statement has no effect on a region with the **no scale** attribute specified.

Examples: response to NewAlp
 change Alpha window size by 50 50

 response to NewGraph
 change Graph in Display window size by -20 10

 action Changer is
 change Box window size to 30 columns 50 lines

See Also: **change size** Action Statement

Character Set

Remarks: EASEL OS/2 EE statements can appear in *free format*. The specific format of words in a line, or lines on a page, is not significant. Wherever a space can appear in a program, you can specify multiple spaces, or you can skip to a new line. The only exceptions to this rule are within string literals or comments.

The EASEL OS/2 EE character set consists of the following ASCII characters:

Character	Meaning	Usage
SPACE	Blank	Throughout EASEL OS/2 EE
A-Z	Uppercase letters	Throughout EASEL OS/2 EE
a-z	Lowercase letters	Throughout EASEL OS/2 EE
0-9	Digits	Throughout EASEL OS/2 EE
=	Equal sign	Relational operator
+	Plus sign	Arithmetic operator
-	Minus sign	Arithmetic operator
*	Multiplication sign	Arithmetic operator
/	Division sign	Arithmetic operator
\	Backslash	EASEL OS/2 EE escape character
(	Left parenthesis	Required in expressions, subroutines, and functions
)	Right parenthesis	Required in expressions, subroutines, and functions
[	Left bracket	Required in arrays
]	Right bracket	Required in arrays
#	Pound sign	Comment character
,	Comma	Separator in argument lists and drawing statements
.	Decimal point	Floating point values
<	Less than	Relational operator
>	Greater than	Relational operator
"	Quotation mark	String delimiter
_	Underscore	Allowed in identifiers
:	Colon	Separator in argument lists and pattern scaling
All characters		String values

See Also: **COMPILE.CMD** OS/2 Command File (Chapter 3)
EASEL.CMD OS/2 Command File (Chapter 3)

check and uncheck Action Statements

Purpose: To check or uncheck a radio button, check box, or choice item.

Model #1 (for Radio Buttons and Check Boxes):

{ check | uncheck } OBJECT_NAME

Model #2 (for Choices):

{ check | uncheck } item
{ ITEM_NAME [from AB_NAME] | ITEM_CLASS_NAME }
in { REGION_NAME | REG_CLASS_NAME }

Where: OBJECT_NAME is the identifier for a radio button or check box. If a radio button is checked, the center is filled in. If a check box is checked, the box is filled with an X.

ITEM_NAME is the identifier for a choice. If checked, a choice is displayed with a checkmark to its left.

AB_NAME is the identifier for an action bar template.

ITEM_CLASS_NAME is the identifier for an item class containing a group of items to be checked or unchecked.

REGION_NAME is the identifier for a region containing the item.

REG_CLASS_NAME is the identifier for a class of regions.

Remarks: When checking a radio button that is a member of a group, all other radio buttons in the group will be unchecked.

You must specify an AB_NAME if the name ITEM_NAME appears in more than one template.

Only the items specified in region REGION_NAME are checked or unchecked. If you specify REG_CLASS_NAME, the specified items from the regions in the class are checked or unchecked.

Examples: response to Copy_To_All
 check Copy_To_All
 check Copy_To_Disk
 check Copy_To_Network

 response to item Bar from View_Graph_Window_AB
 check item Bar from View_Graph_Window_AB
 in View_Graph_Window
 uncheck item Pie from View_Graph_Window_AB
 in View_Graph_Window
 uncheck item Line from View_Graph_Window_AB
 in View_Graph_Window

See Also: **choice** Definition
 is checked Object/Item Inquiry Built-in Function

check box Definition

Purpose: To define a check box.

Model

[enabled | disabled] [visible | invisible] [checked]
check box CB_NAME
size WIDTH HEIGHT
at [position] X Y
in DB_NAME
[parameter is PAR_STRING]
[group is GROUP_NAME]
[text TEXT_STRING]

Where: CB_NAME is the identifier for a check box.

WIDTH and HEIGHT are integer values representing the dimensions of the check box in dialog units.

X and Y are integer values representing the position of the check box in dialog units.

DB_NAME is the identifier for the dialog box or dialog region parent object.

PAR_STRING is a string value representing the parameter of the check box.

GROUP_NAME is an identifier for the group that this check box belongs to.

TEXT_STRING is a string value representing the text of the check box.

Use of **checked**

Keywords: Specifies that this check box is to be displayed checked.

group is ...

Specifies that this check box is to be grouped with all other dialog control objects that specify this group.

text ...

Specifies the text associated with the check box. This text will be displayed to the right of the check box. If a character in the string is preceded by the ~ character, that character is used as the mnemonic for the check box and will be underlined in the text of the check box.

Remarks: A check box is a dialog control, and it must have a dialog box or dialog region as its parent.

Only the **check box** keywords, the identifier, the **at** [**position**] specification, the **in** DB_NAME, and the **size** specification are required in the definition.

When specifying attributes, **parameter**, **group is**, and **text** can be specified in any order; all other attributes must be specified in the order shown in the model.

Default	SELECTABILITY:	enabled
Values:	VISIBILITY:	visible
	PARAMETER:	""
	CHECKED:	unchecked
	TEXT:	""

Examples: invisible dialog box SaveFile
size 160 100 at position 200 200
dialog border
title bar "Save Output File"
system menu

check box Verify
size 48 11
at position 6 20 in SaveFile
text "Verify after write"

See Also: **is checked** Object/Item Inquiry Built-in Function

checked

checked Object/Item Inquiry Built-in Function

See: **is checked** Object/Item Inquiry Built-in Function on page 1-190

checked in group Object Inquiry Built-in Function

Purpose: To find out which radio button in a group is checked.

Model

checked in group RBG_NAME **from** DB_NAME

Where: RBG_NAME is the identifier for a radio button group.
DB_NAME is the identifier for the dialog box in which
RBG_NAME is defined.

Remarks: The value returned is a string that will contain the object
name of the radio button currently checked in the specified
group.

Examples: response to OK
copy checked in group Graph_Types from Graph_DB to
Graph_Name

See Also: **check** and **uncheck** Action Statements
is checked Object/Item Inquiry Built-in Function

choice Definition

Purpose: To define a pulldown choice item.

Model

```
[ enabled | disabled ] [checked] [static]
choice CHOICE_NAME text CHOICE_TEXT
    [accelerator is [ { control | alt | shift }
                    [ control | alt | shift ]... ] ACCEL_KEY]
    [parameter is PAR_STRING]
```

Where: CHOICE_NAME is the identifier for the choice.

CHOICE_TEXT is a string value representing the text that should appear in the choice. This text can be any string. The text can contain the character ~ to indicate the mnemonic character. The text can also include the following escape characters:

\a right aligns the choice text following the escape sequence

\t left aligns a second column of text following the escape sequence.

Note: You cannot use \t with a static choice.

ACCEL_KEY is an integer value selected from a predefined list found in an include file (**accel.inc**), representing the associated accelerator key. The **accel.inc** file assigns integer constant names to each of the integer values, and these constant names can be used if this file is included. You can combine the accelerator key with any number of the status flags **control**, **alt**, or **shift**.

PAR_STRING is a string value representing the parameter of the choice.

Use of **disabled ...**
Keywords: Specifies that the choice is displayed in halftone and cannot be selected.

checked ...
Specifies that the choice is displayed with a checkmark to its left.

static ...

Specifies that the choice is for informational purposes only and cannot be selected or moved to.

text ...

Specifies the text associated with the choice. This text will be displayed in the pulldown. If a character in the string is preceded by the ~ character, that character is used as the mnemonic for the choice and will be underlined in the text of the choice.

accelerator is ...

Specifies the accelerator key that the user can press to select the choice.

... control ...

Specifies that the user must hold down the Control key when pressing the key defined in ACCEL_KEY.

... alt ...

Specifies that the user must hold down the Alt key when pressing the key defined in ACCEL_KEY.

... shift ...

Specifies that the user must hold down the Shift key when pressing the key defined in ACCEL_KEY. The ACCEL_KEY character string is case-independent. It is the **shift** keyword that determines that the accelerator key will be uppercase.

Remarks: A choice is one of the units in a pulldown. It is selectable if the choice is enabled and not static.

Be sure to identify the accelerator keys in the text of the pulldown choice, so that the user knows which keys to press.

To use the constant names in the **accel.inc** file, be sure to include the file. (See “accel.inc File” on page 4-3 for a list of the valid names.)

The parameter of an item is attached to, and is an attribute of, the item defined in the action bar template or the stand-alone item template. If the parameter is changed in an action statement, it is changed in the template definition and everywhere it is referenced.

Choice items can be defined within a pulldown or as a stand-alone template. As a stand-alone template, it can be referenced (and therefore included) in any pulldown definition.

choice

Examples: include accel.inc

```
action bar Edit_Mail_AB is
  pulldown Edit text "~Edit"
    choice Undo text "~Undo\tAlt+Backspace"
      accelerator is alt KEY_BACKSPACE
    separator
    choice Cut text "Cu~t\tShift+Del"
      accelerator is shift KEY_DELETE
    choice Copy text "~Copy\tCtrl+Ins"
      accelerator is control KEY_INSERT
    choice Paste text "~Paste\tShift+Ins"
      accelerator is shift KEY_INSERT
    separator
    choice Clear text "Cl~ear\tDel"
      accelerator is KEY_DELETE
  end pulldown
end action bar
```

See Also: **accel.inc** File (Chapter 4)
 action bar is Template Definition
 item Reference
 pulldown Definition

circle Drawing Statement

Purpose: To draw a circle in a graphical object.

Model

[solid] [**[color]** COLOR] **circle** **[radius]** RADIUS

Where: COLOR is a string or integer value representing a color.
RADIUS is an integer value representing the radius of the circle.

Use of Keywords: **solid ...**
Fills the circle with the circle's color.

[color] COLOR ...
Draws the circle in the specified color, overriding all other color specifications that otherwise would affect this drawing statement.

Remarks: EASEL OS/2 EE automatically draws the circle centered at the current position of the graphics cursor. The final position of the graphics cursor is the same as its initial position, and the text cursor is moved to that position.

Default Values: If **solid** is not specified, just the circle's perimeter (outline) will be drawn.

If color is not specified, the circle will be drawn in the color specified in a pattern reference or object definition.

Examples: response to NewScreen
add to Start
solid color NewColor circle radius Dist

class

class Definition

Purpose: To group a number of objects under one identifier. This allows you to specify one action statement or one response definition for a group of objects.

Model

```
class CLASS_NAME [is  
    OBJECT_NAME1 [OBJECT_NAME2]... ]
```

Where: CLASS_NAME is the identifier for a class of objects.
OBJECT_NAME1 and OBJECT_NAME2 are identifiers for objects.

Remarks: All of the objects in the **class** definition must have been previously defined in the program. The class can be initially empty.

As a general rule, it is inadvisable to group different types of objects as members of the same class. You should define classes that consist of only one type of object.

You cannot define a class as a member of another class; only individual objects can be defined as class members.

An object can be a member of more than one class.

Once a class is defined, you can reference it in any action statement that is valid for an object. When the actions are executed, they will affect all of the objects in the class. Also, you can define a response to the class, and the response will be invoked by any of the objects in the class. Examples are as follows.

Examples:

```
class Invalid_Codes

class Keys is
    KeyA KeyB KeyC ClearKey PanicKey
    :
response to Keys      # If KeyA, KeyB, KeyC, ClearKey,
    make Panel visible # or PanicKey is selected, Panel
                        # becomes visible.
```



```
response to GoBack      # When GoBack is selected, all of  
    make Keys invisible # the objects in the class Keys  
                        # are made invisible.
```

See Also:

add to class Action Statement

delete from class Action Statement

members of Special Inquiry Built-in Function

clear Action Statement

Purpose: To delete the contents of an object.

Model #1 (for Graphical Regions, Image Regions, Textual Regions, and Keys):

clear [graphics] { OBJECT_NAME | CLASS_NAME }

Model #2 (for Dialog Boxes and Dialog Controls):

clear { OBJECT_NAME | CLASS_NAME }

Where: OBJECT_NAME is the identifier for an object.
CLASS_NAME is the identifier for a class of objects.

Use of **clear graphics ...**
Keywords: Deletes the contents of an object, but does not delete, or affect in any way, any of the object's children. Thus, you need not first delete the object's children from any classes to which they belong.

Remarks: If the keyword **graphics** is omitted in Model #1 (for graphical regions, image regions, textual regions, and keys), EASEL OS/2 EE deletes the existing drawing statements for the object and deletes all its children.

Since a textual region's drawing statements are distinct from graphical drawing statements, do not specify a **clear** statement for a class of objects that includes textual regions as well as graphical objects.

Before you delete the contents of an object, it is recommended that you first delete its children from any classes to which they belong, using the **delete from class** action statement, unless the **graphics** keyword is used.

Clearing the contents of push buttons, radio buttons, check boxes, list boxes, entry fields, group boxes, and static text will clear only the text associated with the object and not the graphics.

Clearing the contents of a dialog box will delete all the dialog controls that are its children.

Examples: response to Undo
 clear StopKey

 response to Send
 clear Start in Box

See Also: **change** Action Statement
 delete from class Action Statement

clipboard Special Built-in Function

Purpose: To retrieve textual data from the Presentation Manager clipboard.

Model

clipboard

Remarks: The **clipboard** function returns a string containing the text currently in the clipboard. This function does not affect the contents of the clipboard.

Although the clipboard can simultaneously contain different types of data, this function can only be used to retrieve text. If no text exists in the clipboard, a null string is returned.

Examples: # Copy clipboard contents to a textual region.
add to PasteRegion
insert clipboard

See Also: **copy** Action Statement

close Action Statement

See: **open** and **close** Action Statement on page 1-240.

color Keyword

Purpose: To specify the color used for the foreground of a key, the foreground, background, and/or border of a graphical, image, or textual region, the foreground and/or background of text in a **colored textual region**, or the color of a drawing statement.

Model

[color] COLOR

Where: COLOR is a string value representing a color keyword, or is an integer value representing a color number corresponding to a color available on your system. Refer to the appropriate object definition, action statement, or drawing statement for the proper format within which the COLOR specification must appear.

Remarks: If you use an integer value or a string value to describe a color, you must precede it with the keyword **color**. It is only optional if you specify a color keyword, without quotation marks.

Valid string values are the color keywords listed below and enclosed in quotation marks.

Standard None of these values can be changed.

EASEL
OS/2 EE
Color Set:

Color Keyword	Color Number
red	1
green	2
yellow	3
blue	4
magenta	5
purple	5
cyan	6
aqua	6
white	7
black	8

The specific set of color numbers available to you is dependent on the computer's graphics display adapter.

*Additional
EASEL
OS/2 EE
Colors:* Colors 18-25 can be changed for fullscreen programs.

Color Number	Windowed Color	Fullscreen Default Color (EGA/VGA Value)
18	dark gray	light gray (7)
19	dark blue	dark gray (56)
20	dark red	light blue (11)
21	dark purple	dark purple (5)
22	dark green	bright pink (37)
23	turquoise	orange (52)
24	mustard	gold (38)
25	light gray	brown (20)
26	window background	window background
27	window text	window text

Colors 26 and 27 are set in the PM Control Panel. These colors cannot be used in a **colored textual region**.

You can change colors 18-25 for fullscreen programs by using the **EslSetEgaColor()** function.

*Default
Values:* If no color is specified for a drawing statement, the drawing statement uses the foreground color specified in the object definition.

If no color is specified in the object definition, the following default colors are used in windowed and fullscreen programs:

Graphical, Image,
and Textual Region: **white** foreground
 black background
 white border

Key: **white**

Examples: disabled yellow key Box at position 50 50
 blue box 25 25
 move to 100 100
 circle radius 20

color ColorChoice textual region Blackboard
 window size 30 30
 at position 100 100

color 15 key Help at position 45 30

color

See Also: **EslQueryEgaColor()** Function (Chapter 2)
 EslSetEgaColor() Function (Chapter 2)
 graphical region Definition
 image region Definition
 key Definition
 make COLOR Action Statement
 textual region Definition

column size of Object Inquiry Built-in Function

Purpose: To find the width of a column in a specified textual region.

Model

column size of TR_NAME

Where: TR_NAME is the identifier for a textual region.

Remarks: **column size of** is an integer function, returning the width of a column. This is the width of a character plus the character's associated horizontal intercell gap. The width is in the parent's coordinates.

Examples: response to Changer
change Logo in CoData position to
((xcursor of Types) * (column size of Types)) 100

See Also: **font Reference**
line size of Object Inquiry Built-in Function

combination box Definition

Purpose: To define a combination box.

Model

```
[ enabled | disabled ] [ visible | invisible ]  
[dropdown] combination box CB_NAME  
size WIDTH HEIGHT  
at [position] X Y  
in DB_NAME  
[parameter is PAR_STRING]  
[text size TEXT_LIMIT columns]  
[sort { ascending | descending } ]  
[group is GROUP_NAME]  
[text EF_STRING]  
[insert { LB_STRING | file ASCII_FILE_NAME } ]...
```

Where: CB_NAME is the identifier for the combination box.

WIDTH and HEIGHT are integer values representing the dimensions of the combination box in dialog units.

X and Y are integer values representing the position of the combination box in dialog units.

DB_NAME is the identifier for the dialog box or dialog region parent object.

PAR_STRING is a string value representing the parameter of the combination box.

TEXT_LIMIT is an integer value representing the maximum number of characters that can be entered into the entry field portion of the combination box.

GROUP_NAME is a string value identifying the group to which the combination box belongs.

EF_STRING is a string value representing the initial value that the entry field portion of the combination box will contain.

LB_STRING is a string value to be inserted into the list box portion of the combination box.

ASCII_FILE_NAME is a string value representing the name, including the extension, of an ASCII file to be inserted into the list box portion of the combination box.

*Use of
Keywords:*

dropdown

Specifies that this will be a drop-down combination box.

text size ... columns

Specifies the maximum number of characters that can be entered into the entry field portion of the combination box.

sort ...

Specifies that entries in the list box portion of the combination box are to be sorted in ascending or descending order. If sort is not specified, entries inserted into the list box will be added at the end.

group is ...

Specifies that this combination box is to be grouped with all other dialog control objects that specify this group name.

text ...

Specifies the initial text that the entry field portion of the combination box will contain.

insert ...

Specifies text strings or ASCII files to be inserted into the list box portion of the combination box.

Remarks:

A combination box is a dialog control and must have a dialog box or dialog region as its parent.

Only the **combination box** keywords, the identifier, the **at [position]** specification, the **in DB_NAME**, and the **size** specification are required in the definition.

When specifying attributes, the **parameter**, **text size**, **sort**, **group is**, and **text** can be specified in any order; all other attributes must be specified in the order shown in the model.

Since a combination box contains both list box and entry field functionality, the operations and built-in functions available for these two control objects are also available for a combination box.

*Default
Values:*

SELECTABILITY:	enabled
VISIBILITY:	visible
PARAMETER:	""
TEXT SIZE:	32
TEXT:	""

combination box

Examples: dropdown combination box ColorChoice
 size 80 50 at 50 40 in DBName
 insert "Blue\nGreen\nRed"

See Also: **dropdown list** Definition

Comments

Purpose: To annotate a program.

Model

TEXT

Where: TEXT is a comment string containing any ASCII characters.

Remarks: Comments are ignored by the EASEL OS/2 EE processor. Any text from the comment character (#) to the end of the line is considered part of the comment, even quotation marks and additional comment characters. A comment character always begins a comment except when it is included in a quoted string.

Examples: # Program interfaces to FORTRAN application.

box 50 50 # This draws a box

string variable X is " # not a comment " # But this is.

constant Definition

Purpose: To define a constant, which is a named entity that holds a data value that does not change during execution. Once defined, it can be referenced in the program.

Model

```
[ integer | float | string | boolean ] constant  
  CON1 is VALUE1  
  [CON2 is VALUE2]...
```

Where: CON1 and CON2 are identifiers representing constant names.

VALUE1 and VALUE2 are compile-time values of the same type as their associated constant, representing the values of the constants.

Remarks: The **constant** statement defines one or more constants. A constant must be defined before it is referenced. You can use a constant wherever a value or a compile-time value is required.

When you define a constant, you define its type. The type that you specify for a constant determines the type of value required for the initialization.

Default Values: If the type is not specified, it defaults to **string**.

Examples: float constant Maximum is 24.5

```
constant Limit is "5000"    # Both are string  
      Trigger is "OK"      # constants.
```

copy Action Statement

Purpose: To assign one or more values to a variable or the clipboard, replacing any existing contents of the variable or text in the clipboard.

Model

```
copy VALUE1 [VALUE2]... to { clipboard | VAR | VAR[ N1[, N2]... ] }
```

Where: VALUE1 and VALUE2 are any values.

VAR is the identifier for a variable.

N1[, N2]... are subscripts for each dimension of an array.

Use of ... clipboard
Keywords: Specifies that VALUE_n is to be copied to the PM clipboard.

Remarks: The existing value of the specified variable is replaced by the new value.

There is no limitation on the source or type of values that you can copy to a variable; for example, you can copy values from expressions, variables, constants, literals, textual regions, built-in functions, and keyboard or application input. If you specify a value of a different type, EASEL OS/2 EE will try to convert the value to the expected type. (See "Type Conversions" on page 1-347.)

When more than one value is copied to a variable, the values are concatenated.

If **clipboard** is specified, after VALUE_n is copied to the clipboard, any textual data that previously existed in the clipboard is entirely replaced by the value of VALUE_n. In this context, "clipboard" is not a built-in function, since it represents the target of a string copy operation and no string value is returned. You cannot append data to the clipboard.

Examples: response to Widget1
 copy Widgets to Inventory

action MoreVals is
 copy (A * 5 - B) to Calc

copy

See Also: **append** Action Statement
 array Definition and Reference
 Type Conversions
 variable Definition

date Special Inquiry Built-in Function

Purpose: To find and use the current date.

Model

date

Remarks: The value returned for the **date** function is the current date as MM/DD/YY, where:

MM is the month, and appears as 1 or 2 digits

DD is the day, and appears as 1 or 2 digits

YY is the year, and appears as 2 digits

To get the date in the format specified in the PM Control Panel or CONFIG.SYS COUNTRY setting, use the **LocalDate()** function.

Examples: response to CheckDate
change Calendar to
date

See Also: **time** Special Inquiry Built-in Function
LocalDate() Function (Chapter 2)

deemphasize Action Statement

Purpose: To remove the emphasis from a text segment in a textual region, or from all text segments in a class of textual regions.

Model

deemphasize { TR_NAME | TR_CLASS_NAME }

Where: TR_NAME is a string value representing the identifier for a textual region.
TR_CLASS_NAME is the identifier for a class of textual regions.

Remarks: This statement can be used only with textual regions.
A text segment is emphasized with the **emphasize** action statement. The **deemphasize** statement removes the emphasis from the segment. Note that you simply specify the textual region name (or class name); you do not specify the actual segment. This is because only one segment at a time can be emphasized in a textual region.

Examples: response to Undo
deemphasize NewText

See Also: **emphasize** Action Statement
segment Definition

delete Action Statement

Purpose: To delete an object.

Model

```
delete { OBJECT_NAME | CLASS_NAME }
```

Where: OBJECT_NAME is the identifier for an object.
CLASS_NAME is the identifier for a class.

Remarks: The **delete** action statement deletes the object itself, its contents, and all of its children.

Before you delete an object, you must first delete the object (as well as its children) from any classes of which it is a member, using the **delete from class** statement. If you do not, performing an action on the class will produce an error during runtime.

Examples: response to ChoiceStart
delete RedKey

response to ChoiceCtd
delete BlueKey in Panel

See Also: **clear** Action Statement
delete from class Action Statement

delete action bar from Action Statement

Purpose: To delete an action bar from a region or from all the regions in a class.

Model

delete action bar from { REGION_NAME | REG_CLASS_NAME }

Where: REGION_NAME is the identifier for a region (graphical, image, or textual).
REG_CLASS_NAME is the identifier for a class of regions.

Examples: response to item DeleteActionBar
delete action bar from MainRegion

response to SeeMore
delete action bar from AllRegions

delete from class Action Statement

Purpose: To remove an object, or a class of objects, from a class.

Model

```
delete { OBJECT_NAME | CLASS_NAME } from class CLASS_NAME
```

Where: OBJECT_NAME is the identifier for an object.
CLASS_NAME is the identifier for a class of objects.

Remarks: Deleting an object, or class of objects, from a class has no effect on the existence or attributes of the object itself; the object still exists. The only results of deleting an object from a class are that: (1) selecting the specified object no longer invokes the responses that were defined for the class, and (2) any action statements that reference the class no longer affect that object.

When an object is a member of more than one class, deleting it from one class has no effect on its membership in other classes.

Examples: response to Omit
delete Rolls from class Luxury_cars

response to Clear
delete Fuels in Expenses from class Heating_costs

See Also: **class** Definition
members of Special Inquiry Built-in Function

delete from item class Action Statement

Model

```
delete item { ITEM_NAME [from AB_NAME] | ITEM_CLASS_NAME }  
  from item class ITEM_CLASS_NAME
```

Where: ITEM_NAME is the identifier for a pulldown, button, choice, or separator.

AB_NAME is the identifier for an action bar template. The action bar name must be specified if more than one template contains an item of the given name.

ITEM_CLASS_NAME is the identifier for a class containing items.

Remarks: Deleting an item, or a class of items, from a class has no effect on the existence or attributes of the item itself; the item still exists. The only results of deleting an item from a class are that (a) selecting the specified item no longer invokes the responses that were defined for the class, and (b) any action statements that reference the class no longer affect that item.

When an item is a member of more than one class, deleting it from one class has no effect on its membership in other classes.

If **from** AB_NAME was specified when the item was added to the item class, then **from** AB_NAME must be specified in the **delete item** statement. If **from** AB_NAME was not specified when the item was added to the item class, then **from** AB_NAME should not be specified in the **delete item** statement.

Examples: response to DeleteIt
 delete item MoreInfo from item class Information

 response to item CreateKeys
 delete item object from item class CreateKeys

delete item Action Statement

Purpose: To delete an item from a region or class of regions.

Model

```
delete item { ITEM_NAME [from AB_NAME] | ITEM_CLASS_NAME }
in { REGION_NAME | REG_CLASS_NAME }
```

Where: ITEM_NAME is the identifier for a pulldown, button, choice, or separator.

AB_NAME is the identifier for an action bar template.

ITEM_CLASS_NAME is the identifier for an item class containing a group of items to be deleted.

REGION_NAME is the identifier for the region containing the item.

REG_CLASS_NAME is the identifier for a class of regions.

Remarks: You must specify an AB_NAME if the name ITEM_NAME appears in more than one template.

Only the items specified in region REGION_NAME are deleted. If you specify REG_CLASS_NAME, the specified items from the regions in the class are deleted.

Examples: response to DeleteIt
 delete item MoreInfo from MainAction in MainRegion

response to item CreateKeys
 delete item object in MainRegion

deselect Action Statement

See: **select** and **deselect** Action Statements on page 1-302.

dialog box Definition

Purpose: To define a dialog box.

Model

```
[ enabled | disabled ] [ visible | invisible ]
[ modal | modeless ] dialog box DB_NAME
  size WIDTH HEIGHT
  at [position] X Y
  [parameter is PAR_STRING]
  [ [dialog] border ]
  [title bar TITLE_STRING]
  [system menu]
```

Where: DB_NAME is the identifier for a dialog box.

WIDTH and HEIGHT are integer values representing the dimensions of the viewport, in dialog units.

X and Y are integer values representing the position of the dialog box in dialog units.

PAR_STRING is a string value representing the parameter of the dialog box.

TITLE_STRING is a string value representing the text that will appear in the title bar of the dialog box.

*Use of
Keywords:*

modal

Specifies that the dialog box will be modal. When the box is made visible, objects outside of the box cannot be selected until the box is made invisible or deleted.

modeless

Specifies that the dialog box will be modeless. This allows the user to interact with visible objects outside of the dialog box, while the dialog box is still visible.

... border

Specifies whether or not the dialog box will have a border drawn around it. If only **border** is specified, the standard PM window border will be drawn. If **dialog border** is specified, the standard PM dialog border will be drawn.

dialog box

title bar

Specifies whether the dialog box will contain a title bar. If **title bar** is specified, the standard PM title bar will be added to the dialog box.

system menu

Specifies whether the dialog box will contain the standard PM system menu icon.

Remarks: Only the **dialog box** keywords, the identifier, the **at** [**position**] specification, and the **size** specifications are required in the definition.

When specifying attributes, the **parameter**, **border**, **title bar**, and **system menu** can be specified in any order; all other attributes must be specified in the order shown in the model.

Dialog boxes can have only dialog control objects as children.

Pressing the ESCAPE key, double-clicking on the system menu, or selecting the "Close" choice from the system menu invokes a response to the cancel push button, if there is one.

The ancestry of a dialog box is always **in desktop**.

<i>Default</i>	SELECTABILITY:	enabled
<i>Values:</i>	VISIBILITY:	visible
	MODE:	modal
	ANCESTRY:	in desktop
	PARAMETER:	""
	BORDER:	none
	TITLE BAR:	none
	SYSTEM MENU:	none

Examples: invisible dialog box SaveFile
size 160 100 at position 200 200
dialog border
title bar "Save Output File"
system menu

Dialog Control Objects

Remarks: Dialog controls are objects that exist in dialog boxes and dialog regions. The following objects are dialog controls:

check box
combination box (simple and dropdown)
dropdown list
entry field (single-line)
group box
list box
multiline entry field
push button
radio button
static text

See Also: **dialog box** Definition
dialog region Definition

dialog region Definition

Purpose: To define a dialog region, which is a rectangular portion of the screen in which dialog controls can be placed.

Model

```
[ enabled | disabled ] [ visible | invisible ]  
[primary] dialog region DR_NAME  
  size XSIZE YSIZE  
  at { [position] X Y | default position }  
  [in PARENT_NAME1 [in PARENT_NAME2]... ]  
  [priority is PRIORITY_NO]  
  [parameter is PAR_STRING]  
  [ [size] border]  
  [title bar TITLE_STRING]  
  [system menu]  
  [horizontal scroll bar [scroll by SCROLL_INC] ]  
  [vertical scroll bar [scroll by SCROLL_INC] ]  
  [action bar AB_NAME]  
  [minimize button [using MINIMIZE_ICON_FILE] ]  
  [maximize button]
```

Where: DR_NAME is the identifier for the dialog region.

XSIZE and YSIZE are integer values representing the dimensions of the region's viewport in dialog units, relative to the origin of the parent's coordinate system.

X and Y are integer values representing the position of the viewport in dialog units, relative to the origin of the parent's coordinate system (not the position of the outside frame).

PARENT_NAME1 and PARENT_NAME2 are identifiers for parent objects. If it is a primary region, you cannot specify ancestry; it is always **desktop**. To place a region outside of a primary region, specify its ancestry as **in desktop**.

PRIORITY_NO is an integer value representing the priority of the region. The primary region and regions whose ancestry is desktop cannot be assigned a priority.

PAR_STRING is a string value representing the parameter of the region.

TITLE_STRING is a string value representing the title that will appear in the region's title bar.

SCROLL_INC is an integer value representing the number of units in the region's coordinate system by which the region's window position will be changed when one of the arrows in the scroll bar is pressed. When paging, the page increment is the X or Y window size minus one scroll increment.

AB_NAME is the identifier of the action bar template.

MINIMIZE_ICON_FILE is a string value representing the name, including the extension, of the file containing the icon to use when the region is minimized.

*Use of
Keyword:*

primary

Specifies that this region is the program's primary region.

size ...

Defines the size of the region's viewport in dialog units.

... border

Specifies the type of border present around the region. If **border** is specified, a one-pixel border will be drawn.

If **size border** is specified, the region will be surrounded by a sizing border that can be used to resize the region.

title bar ...

Specifies that the region will contain a title bar. This will generate a title bar that can be used for repositioning the region. The title bar will contain the text specified.

system menu

Specifies that the region will contain a system menu. The system menu button will be in the upper left corner.

horizontal scroll bar ...

Specifies that the region will contain a horizontal scroll bar. The scroll bar will be located along the bottom edge of the region just inside the border, if one exists. The region's contents will be scrolled as specified by **SCROLL_INC**.

vertical scroll bar ...

Specifies that the region will contain a vertical scroll bar. The scroll bar will be located along the far right edge of the region just inside the border, if one exists. The region's contents will be scrolled as specified by **SCROLL_INC**.

action bar ...

Specifies that the region will contain an action bar.

dialog region

minimize button ...

Specifies that the region will contain a minimize button in the upper right corner.

maximize button

Specifies that the region will contain a maximize button in the upper right corner. (The maximize button will turn into a restore button when the region is maximized.)

Remarks:

In a windowed application, if the dialog region is the primary region, it must be the first object you define. There can be only one primary region.

The coordinates used in the definition of a dialog region (and therefore its children) are dialog units.

Dialog regions can have only dialog controls as children.

Dialog regions always resize by clipping rather than by scaling.

Only the **dialog region** specification, the identifier, the **at [position]** specification, and the **size** specifications are required, and they must be specified in the order shown in the model. Other attributes are optional, using the defaults shown below.

Any of the attributes after ancestry can be specified in any order.

The region size only includes the area of the viewport itself. It does not include the frame attributes, which are external to the viewport. If the size of the primary dialog region is the same as the screen size, you will not be able to see any frame attributes.

Pressing the ESCAPE key, double-clicking on the system menu, or selecting the "Close" choice from the system menu invokes a response to the cancel push button, if there is one. If no response to the cancel push button is found, EASEL OS/2 EE looks for and invokes an **on close** response for that region. If there is no cancel push button and no **on close** response to invoke, EASEL OS/2 EE destroys the dialog region.

In a windowed application, disabling a primary dialog region disables its viewport and its frame, and also disables the dialog region's children.

In a fullscreen application, a dialog region is disabled by default, and a dialog region's selectability does not affect the selectability of its children.

Default **SELECTABILITY:** **enabled**
Values: **VISIBILITY:** **visible**
 ANCESTRY (primary): **in desktop**
 ANCESTRY: **in primary region**
 PRIORITY: **0**
 PARAMETER: **""**
 BORDER: invisible
 SCROLL_INC: one dialog unit

Examples: primary dialog region EntryScreen
 size 200 160 at 100 100
 system menu
 title bar "Shipping Information System"
 border

disable

disable Action Statement

See: **enable** and **disable** Action Statements on page 1-119.

disabled Attribute Definitions

See: **enabled** and **disabled** Attribute Definitions on page 1-121.

draw Drawing Statement

Purpose: To draw a line in a graphical object.

Model

```
[ [color] COLOR] draw
{ [by] XOFFSET YOFFSET[, XOFFSET YOFFSET]... | to X Y[, X Y]... }
```

Where: COLOR is an integer or string value representing an EASEL OS/2 EE color.

XOFFSET and YOFFSET are integer values representing the relative offsets of the end of a line.

X and Y are integer values representing the absolute coordinate positions of the end of a line.

Use of [color] COLOR ...
Keywords: Draws the line (or sets of lines) in the specified color, overriding all other color specifications that otherwise would affect this drawing statement.

... [by] XOFFSET YOFFSET
Draws a line from the current position to a new position that is offset from the current position by XOFFSET and YOFFSET.

... to X Y
Draws a line from the current position to a new absolute position specified by integer values representing the X and Y coordinates in the coordinate system of the object.

Remarks: EASEL OS/2 EE begins at the current position of the graphics cursor. The **draw** statement moves the graphics and text cursors exactly as the **move** statement does, but it also draws a line from the old position to the new.

Any number of sets of coordinates can be specified in a **draw** statement; EASEL OS/2 EE will continue to move the cursor and draw additional lines. Each set of X and Y coordinates must be separated by commas.

You must specify a separate **draw** statement for each different color. See the example of the Blackboard graphical region below.

Default If color is not specified, the line (or sets of lines) will be
Values: drawn in the color specified in a pattern reference or
object definition.

Examples: key Redo at position 50 25
draw 20 40 # The line is white.

response to Redo
add to Redo
red draw 50 50, 50 0, 50 -50 # The lines are red.

graphical region Blackboard size 100 100 at position 10 5
blue foreground
draw to 100 150 # This line is blue.
The following line is drawn in the color stored in
the Color_A variable:
color Color_A draw to 25 25, 100 100

See Also: **draw arc** Drawing Statement
move Drawing Statement

draw arc

draw arc Drawing Statement

Purpose: To draw an arc as part of a boundary specification within a shape definition.

Model #1 (In Specified Degrees):

```
draw arc DEGREES degrees [ counter clockwise | clockwise ]
      center [ by | to | at ] XCENTER YCENTER
```

Model #2 (To a Specified Endpoint):

```
draw arc { by | to } XEND YEND [ counter clockwise | clockwise ]
      center [ by | to | at ] XCENTER YCENTER
```

Where: DEGREES is an integer value representing the number of degrees of the arc.

XCENTER is an integer value representing the horizontal center of the arc.

YCENTER is an integer value representing the vertical center of the arc.

XEND is an integer value representing the horizontal endpoint of the arc.

YEND is an integer value representing the vertical endpoint of the arc.

Use of Keywords:

- ... **DEGREES degrees**
Draws an arc of the specified number of degrees. Positive degrees draws the arc in the direction of travel around the circle. Negative degrees draws the arc opposite to the direction of travel around the circle.
- ... **counter clockwise**
Defines the direction of travel around the circle to be counter clockwise. This is the default.
- ... **clockwise**
Defines the direction of travel around the circle to be clockwise.

... center by

Specifies the center of the arc as a position relative to the beginning of the arc, which is at the current position of the graphics cursor.

... center to or center at

Specifies the center of the arc as an absolute position in the coordinate system of the object. The keywords **to** and **at** are synonymous.

... by XEND YEND

Draws an arc by the specified number of coordinate positions relative to the current position of the graphics cursor. If the starting and ending points of the arc are not the same distance from the center and a **center** is specified, the arc is drawn with the starting point and the center exactly as specified, and the endpoint of the arc is drawn as close as possible to the specified endpoint, while preserving the circularity of the arc.

... to XEND YEND

Draws an arc to the absolute coordinate position specified by XEND and YEND. If the starting and ending points of the arc are not the same distance from the center and a **center** is specified, the arc is drawn with the starting point and the center exactly as specified, and the endpoint of the arc is drawn as close as possible to the specified endpoint, while preserving the circularity of the arc.

Remarks:

The **draw arc** statement can only be used within a **boundary** specification within a **shape** drawing statement. This statement draws a specified portion of a circle; all points along the arc are equidistant from the center point.

When EASEL OS/2 EE can draw a straight line between the starting point and the endpoint of the arc, a single **draw arc** statement is a complete boundary specification.

A negative arc drawn when the direction of travel around the circle is counter clockwise produces the same result as a positive arc drawn when the direction of travel is clockwise.

draw arc

The **center** specification defines the center of the circle of which the arc is a part. You must provide a **center** specification for the first arc in the first **shape** statement in a graphical object. Thereafter, if you specify a series of **move arc** and **draw arc** statements in a single graphical object, and if each arc has the same center, you do not need to provide additional **center** specifications. If you do not specify any **center** specification in one of these subsequent **move arc** or **draw arc** statements, the center is assumed to be the same as the center of the previous arc in that graphical object.

Default Values: If you omit both the **clockwise** and the **counter clockwise** keywords, the direction of travel around the circle is counter clockwise.

Examples: key Round at position 10 10
draw arc 30 degrees clockwise center at 0 0

graphical region Starter size 300 300 at 0 0
draw arc 180 degrees center XCenter YCenter

See Also: **move arc** Drawing Statement
shape Drawing Statement

dropdown list Definition

Purpose: To define a drop-down list.

Model

```
[ enabled | disabled ] [ visible | invisible ]
dropdown list DL_NAME
  size WIDTH HEIGHT
  at [position] X Y
  in DB_NAME
  [parameter is PAR_STRING]
  [text size TEXT_LIMIT columns]
  [sort { ascending | descending } ]
  [group is GROUP_NAME]
  [text FIELD_STRING]
  [insert { LB_STRING | file ASCII_FILE_NAME } ]...
```

Where: DL_NAME is the identifier for a drop-down list.

WIDTH and HEIGHT are integer values representing the dimensions of the drop-down list in dialog units.

X and Y are integer values representing the position of the drop-down list in dialog units.

DB_NAME is the identifier for the dialog box or dialog region parent object.

PAR_STRING is a string value representing the parameter of the drop-down list.

TEXT_LIMIT is an integer value representing the maximum number of characters that the selection field portion of the drop-down list can contain.

GROUP_NAME is a string value identifying the group to which the drop-down list belongs.

FIELD_STRING is a string value representing the initial value that the selection field portion of the drop-down list will contain.

LB_STRING is a string value to be inserted into the list box portion of the drop-down list.

dropdown list

ASCII_FILE_NAME is a string value representing the name, including the extension, of an ASCII file to be inserted into the list box portion of the drop-down list.

Use of
Keywords:

text size ... columns
Specifies the maximum number of characters that the selection field portion of the drop-down list can contain.

sort ...
Specifies that entries in the list box portion of the drop-down list are to be sorted in ascending or descending order. If sort is not specified, entries inserted into the list box will be added at the end.

group is ...
Specifies that this drop-down list is to be grouped with all other dialog control objects that specify this group name.

text ...
Specifies the initial text that the selection field portion of the drop-down list will contain.

insert ...
Specifies text strings or ASCII files to be inserted into the list box portion of the drop-down list.

Remarks: A drop-down list is a dialog control and must have a dialog box or dialog region as its parent.

Only the **dropdown list** keywords, the identifier, the **at [position]**, specification, the **in DB_NAME**, and the **size** specification are required in the definition.

When specifying attributes, the **parameter**, **text size**, **sort**, **group is**, and **text** can be specified in any order; all other attributes must be specified in the order shown in the model.

The operations and built-in functions available for list boxes and entry fields are also available for a dropdown list.

Default
Values: SELECTABILITY: **enabled**
 VISIBILITY: **visible**
 PARAMETER: **""**
 TEXT SIZE: **32**
 TEXT: **""**

Examples: dropdown list ColorChoice
 size 80 50
 at 50 40 in DBName
 insert "Blue\nGreen\nRed"

See Also: **combination box** Definition
 list box Definition

ellipse Drawing Statement

Purpose: To draw an ellipse in a graphical object.

Model

[solid] [[color] COLOR] ellipse
major [radius] MAJOR minor [radius] MINOR

Where: **COLOR** is a string or integer value representing a color.
MAJOR is an integer value representing the radius of the ellipse in the X direction.
MINOR is an integer value representing the radius of the ellipse in the Y direction.

Use of **solid ...**
Keywords: Fills the ellipse with the ellipse's color.

[color] COLOR ...
Draws the ellipse in the specified color, overriding all other color specifications that otherwise would affect the ellipse.

major radius ...
Specifies the radius parallel to the X axis (horizontal).

... minor radius
Specifies the radius parallel to the Y axis (vertical).

Remarks: EASEL OS/2 EE draws the ellipse centered at the current position of the graphics cursor. The final position of the graphics cursor returns to its initial position, and the text cursor is moved to that position.

Either radius can be larger than the other, or the two radii can be equal. If the two radii are equal, the ellipse becomes a circle.

Default: If **solid** is not specified, only the ellipse's perimeter (outline) will be drawn.

If color is not specified, the ellipse will be drawn in the color specified in a pattern reference, object definition, or other statements which affect the color of this drawing statement.

Examples: response to More
 add to Object
 ellipse major 50 minor 40

 key Help at position 100 100
 solid color NewColor ellipse major 100 minor 150

See Also: **circle** Drawing Statement
 draw Drawing Statement
 draw arc Drawing Statement

emphasize Action Statement

Purpose: To emphasize specified text in a textual region.

Model

emphasize segment [**column**] COL_NO [**line**] LINE_NO
[**thru** [**column**] COL_NO [**line**] LINE_NO] **in** TR_NAME

Where: COL_NO are integer values representing the first and last columns of characters to be emphasized.

LINE_NO are integer values representing the first and last lines of characters to be emphasized.

TR_NAME is the identifier for a textual region.

Remarks: This statement can be used only with textual regions. It cannot be used for a class of textual regions, or for any text block.

Text is emphasized by reversing the foreground and background colors.

Any textual region can have only one segment of emphasized text at any one time. When a second **emphasize** statement is executed for that textual region, the first segment will become deemphasized.

Examples: response to Selected
 emphasize segment column 3 line 1
 thru column 3 line 5 in RegionB

response to Selected
 emphasize segment 4 3 in Blackboard

See Also: **deemphasize** Action Statement
segment Definition

enable and disable Action Statements

Purpose: To change the selectability of an object or item.

Model #1 (for Objects):

```
{ enable | disable } { OBJECT_NAME | CLASS_NAME }
```

Model #2 (for Items):

```
{ enable | disable } item
{ ITEM_NAME [from AB_NAME] | ITEM_CLASS_NAME }
  in { REGION_NAME | REG_CLASS_NAME }
```

Where: OBJECT_NAME is the identifier for an object.

CLASS_NAME is the identifier for a class of objects.

ITEM_NAME is the identifier for a pulldown, choice, or button.

AB_NAME is the identifier for an action bar template.

ITEM_CLASS_NAME is the identifier for an item class containing a group of items to be enabled or disabled.

REGION_NAME is the identifier for the region containing the item.

REG_CLASS_NAME is the identifier for a class of regions.

Remarks: The **enable** keyword makes the object or item selectable if all its ancestors are enabled; the **disable** keyword makes it not selectable.

A push button, radio button, check box, combination box dropdown arrow, dropdown list dropdown arrow, static text, pulldown, button, or choice appears grayed when disabled.

A separator cannot be enabled.

When enabling or disabling an item, you must specify an AB_NAME if the name ITEM_NAME appears in more than one template.

enable and disable

Only the items specified in region `REGION_NAME` are enabled or disabled. If you specify `REG_CLASS_NAME`, the specified items from the regions in the class are enabled or disabled.

Disabling an object implicitly disables its children. Disabling a region with frame attributes also disables the region's frame.

Examples: response to Little
 enable Big

 response to Big
 disable Little in Size

 response to item Open from FileActionBar in MainRegion
 enable item SaveAsIs from FileActionBar
 in MainRegion

 response to item Close from FileActionBar in MainRegion
 disable item FileItems from FileActionBar
 in MainRegion

See Also: **enabled** and **disabled** Attribute Definitions
 selectability of Object Inquiry Built-in Function

enabled and disabled Attribute Definitions

Purpose: To define an object or item as selectable or not selectable.

Model

{ enabled | disabled }

Remarks: Selectability is defined within the object or item definition.

If you define an object or item and all its ancestors as **enabled**, and you have defined a response for that object or item in the program, selecting the object or item causes the actions specified in the response definition to be performed.

If you define an object or item as **disabled**, selecting the object or item does not produce a response, even if you have included a response for that object or item in the program.

An object or item can be selected anywhere in its senseport. The senseport is the smallest rectangular portion of the screen that completely encloses the displayed portion of a selectable object and all of its children, or an item.

A push button, radio button, check box, combination box dropdown arrow, dropdown list dropdown arrow, static text, pulldown, button, or choice appears grayed when disabled.

A separator cannot be enabled.

Default Values: In a windowed application, all objects default to **enabled**.

In a fullscreen application, all regions default to **disabled**.

All items, except for the separator, default to **enabled**.

Examples: enabled textual region Blackboard
 window size 30 columns 50 lines at position 100 100
 disabled key ReturnToMenu at position 10 10

enabled and disabled

See Also:	button Definition	image region Definition
	check box Definition	key Definition
	choice Definition	list box Definition
	combination box Definition	multiline entry field Definition
	dialog box Definition	pulldown Definition
	dialog region Definition	push button Definition
	dropdown list Definition	radio button Definition
	enable and disable Action Statements	static text Definition
	entry field Definition	selectability of Object Inquiry Built-in Function
	graphical region Definition	sense region Definition
	group box Definition	textual region Definition

enabled Item Inquiry Built-in Function

See: **is enabled** Item Inquiry Built-in Function on page 1-192.

entry field Definition

Purpose: To define an entry field.

Model

```
[ enabled | disabled ] [ visible | invisible ]  
[password] entry field EF_NAME  
  size WIDTH HEIGHT  
  at [position] X Y  
  in DB_NAME  
  [parameter is PAR_STRING]  
  [text size TEXT_LIMIT columns]  
  [ { left | right | [horizontal] center } align]  
  [group is GROUP_NAME]  
  [text TEXT_STRING]
```

Where: EF_NAME is the identifier for an entry field.

WIDTH and HEIGHT are integer values representing the dimensions of the entry field in dialog units.

X and Y are integer values representing the position of the entry field in dialog units.

DB_NAME is the identifier for the dialog box or dialog region parent object.

PAR_STRING is a string value representing the parameter of the entry field.

TEXT_LIMIT is an integer value representing the maximum number of characters that can be entered into the entry field.

GROUP_NAME is an identifier for the group that the entry field belongs to.

TEXT_STRING is a string value representing the initial value that the entry field will contain.

Use of

password
Keywords: Specifies that the contents of the entry field will not be visible.

text size ... columns
Specifies the maximum number of characters that can be entered into the entry field.

... align

Specifies the alignment of the text within the entry field.

group is ...

Specifies that this entry field is to be grouped with all other dialog control objects using the specified group name.

text ...

Specifies the initial text or character string that the entry field will contain.

Remarks: An entry field is a dialog control, and it must have a dialog box or dialog region as its parent.

Only the **entry field** keywords, the identifier, the **at [position]** specification, the **in DB_NAME**, and the **size** specification are required in the definition.

When specifying attributes, the **parameter**, **text size**, **align**, **group is**, and **text** can be specified in any order; all other attributes must be specified in the order shown in the model.

Default:

SELECTABILITY:	enabled
VISIBILITY:	visible
METER:	""
TEXT SIZE:	32
ALIGNMENT:	left
TEXT:	""

Examples:

```
invisible dialog box SaveFile
  size 160 100
  at position 200 200
  dialog border
  title bar "Save Output File"
  system menu
```

```
entry field FileName
  size 60 16
  at position 6 20 in SaveFile
  text size 40 columns
  text "file.doc"
```

See Also: **multiline entry field** Definition

errorlevel Action Inquiry Built-in Function

Purpose: To determine whether an external subroutine call, **start** statement or **send** action statement executed successfully.

Model

errorlevel

Remarks: **errorlevel** is an integer function. Its value is 0 if there was no error; otherwise an error code is returned.

Examples: response to start
start local AppName "APP.exe"
if (errorlevel > 0) then
:
end if

See Also: **send** Action Statement
start Action Statement
Chapter 2, Functions and Subroutines

Escape Sequences

Remarks: The table below describes the escape sequences supported by EASEL OS/2 EE. An escape sequence consists of the EASEL OS/2 EE escape character, a backslash (\), followed by a code representing the special character. Escape sequences are used in string literals for those characters that cannot be specified explicitly in the string.

Escape Sequence	Meaning
\"	Quotation mark
\\	Backslash
\a	Right align in action bars
\b (or \^H)	Backspace (destructive)
\e (or \^[)	Escape
\f (or \^L)	Form feed (new page)
\n (or \^J)	Line feed (new line)
\r (or \^M)	Carriage return
\t (or \^I)	Tab (horizontal)
\v (or \^K)	Tab (vertical)

Note: The single character escape sequence \e (or \^[) should be used to specify the ASCII Escape character in a string literal whereas the double character keyboard character code \e\e (or \^[\[) should be used to respond to the Escape (Esc) key typed on the keyboard.

See Also: Keyboard Mapping Table

eventnumber Response Inquiry Built-in Function

See: **response to stimulus** Response Definition on page 1-291.

eventparam Response Inquiry Built-in Function

See: **response to stimulus** Response Definition on page 1-291.

exists Object Inquiry Built-in Function

Purpose: To find out if a specified object currently exists.

Model

exists OBJECT_NAME

Where: OBJECT_NAME is a string value representing the identifier for an object.

Remarks: The value returned is the boolean (**true** or **false**) associated with the existence of the specified object. If the specified object currently exists (it has been defined, and has not been deleted), the value **true** is returned; otherwise, the value **false** is returned.

If you have more than one object with the same name, be sure to specify ancestry with the **exists** function.

Examples: response to Read
 if (exists News in WindowA) then
 :
 end if

exit Action Statement

Purpose: To terminate the current EASEL OS/2 EE program.

Model

exit [EXIT_CODE]

Where: EXIT_CODE is an integer value representing a code to be returned upon program termination.

Remarks: The **exit** statement should be specified as the final statement in the set of action statements in the response definition for terminating the program. Statements occurring after the **exit** statement will not be executed.

Examples: response to ExitKey
 send ShutdownMessage to Audit
 exit

 response to ExitKey
 send ShutdownMessage to Audit
 exit ExitCode

See Also: **change to program** Action Statement

Expressions, Operators, and Operands

Purpose: An expression is a combination of operands and operators that produces a value as a result. An expression can be specified wherever a value is required. Expressions are evaluated in the order (precedence) as listed below.

Operators	Meaning	Operands and Result	Examples
Arithmetic:			
—	(Unary) Minus	integer or float	-5
*	Multiplication	integer or float	(Base * 100)
/	Division	integer or float	(Total / (Unit + 10))
mod	Modulo (remainder of integer division)	integer	(C mod 12)
+	Addition	integer or float	(A + 10)
—	Subtraction	integer or float	((Num2 * 300/A) - 8)
Relational:			
=	Equal to	boolean	(A = "Boston")
!=	Not equal to	boolean	(Slope != 10)
<	Less than	boolean	((B * 50) < C)
< =	Less than or equal to	boolean	(Debts < = Assets)
>	Greater than	boolean	(5000 > Curve)
> =	Greater than or equal to	boolean	(Xtan > = Xsin)
Boolean:			
not	Result is true if operand is false	boolean	(not (A > B))
and	Result is true if both operands are true	boolean	((V1 = 5) and (V2 != 10))
or	Result is true if either or both operand(s) are true	boolean	((Slope < Arc) or (A < = B))

- Remarks:** An expression must be enclosed in parentheses.
- You can use arithmetic, relational, and boolean operators in a single expression. All operands must match the type of operator used.
- If a specified operand type does not match the type of operators, EASEL OS/2 EE attempts to convert the value to the correct type; see "Type Conversions" on page 1-348.
- When operators have equal precedence, the expression is evaluated from left to right.
- You can use nested parentheses to clarify expressions and to override the precedence of operators. If an expression contains nested parentheses, EASEL OS/2 EE always begins with the operation within the innermost set.
- Errors:** Within an expression, the operands for an operator must be of the same type. If they are not, EASEL OS/2 EE generates an error message and does not convert the value.
- See Also:** Type Conversions

extract from Action Statement

Purpose: To extract string values (data fields) from any string value and assign them to a variable.

Model

extract from SOURCE

[SKIP_CLAUSE]...
TAKE_CLAUSE
[TAKE_CLAUSE | SKIP_CLAUSE]...

Where: SOURCE is a string value representing the source of the input being examined. If you are extracting strings from a textual region, the SOURCE should be one of the **textual** built-in functions.

TAKE_CLAUSE is a clause that follows the model:

Model

take [by | **to**] { **marker** | **last** | **word** | **number** |
float | TEXT | COLUMN } VAR_NAME

SKIP_CLAUSE is a clause that follows the model:

Model

skip [by | **to**] { **marker** | **last** | **word** | **number** |
float | TEXT | COLUMN }

Where: TEXT is a string value representing the text being extracted or skipped.

COLUMN is an integer value representing the number of columns (when used with **skip** [**by**] or **take** [**by**]), or the column number (when used with **skip** **to** or **take** **to**).

VAR_NAME is the identifier for a variable name.

*Use of
CLAUSE
Keywords:*

skip to marker and take to marker

Starts with the current pointer position and skips or takes characters up to, but not including, the pointer position at the time of the last **extract from** statement. After execution, the pointer position is the **marker**.

skip to last or take to last

Starts with the current pointer position and skips or takes characters up to and including the last character in the source. After execution, the pointer is positioned after the last character in the source.

skip to word and take to word

Starts with the current pointer position and skips or takes characters up to, but not including, the first printing character encountered. After execution, the pointer is positioned at the beginning of the word.

skip [by] word or take [by] word

Starts with the current pointer position, skips over any leading blanks, and then skips or takes the characters up to but not including the first nonprinting character (e.g., blank) or the end of the source. Numeric characters are considered part of the word. After execution, the pointer is positioned after the last character of the word.

skip [by] number and take [by] number

Starts with the current pointer position, skips over any leading blanks, and then skips or takes subsequent digits (which may include a leading sign) up to the first nondigit (e.g., a blank, a decimal point, or an alphabetic character) or the end of the source. These clauses only skip over spaces and tabs. After execution, the pointer is positioned immediately after the end of the number.

skip to number and take to number

Starts with the current pointer position, and skips or takes blanks, up to the first nonblank character that might start an integer. These clauses only skip over spaces and tabs. After execution, the pointer is positioned at the beginning of the number.

skip [by] float and take [by] float

Starts with the current pointer position, skips over any leading blanks and alphabetic characters, and then skips or takes subsequent digits (which may include a decimal point or a leading sign) up to the first nondigit (e.g., a blank, an alphabetic character, or a second decimal point) or the end of the source. After execution, the pointer is positioned immediately after the end of the floating point number.

skip to float and take to float

Starts with the current pointer position and skips or takes subsequent characters up to the first nonblank, nonalphabetic character that might start a floating point number (e.g., a digit, leading sign, or decimal point). These clauses *do* pass over alphabetic characters. After execution, the pointer is positioned at the beginning of the floating point number.

skip [by] TEXT and take [by] TEXT

Starts with the current pointer position, searches for the beginning of the specified string, and skips or takes the specified string. After execution, the pointer is positioned at the first character after the end of the string.

skip to TEXT and take to TEXT

Starts with the current pointer position, searches for the beginning of the specified string, and skips or takes characters up to but not including the specified string. The pointer is then positioned at the first character in the string.

skip [by] COLUMN and take [by] COLUMN

Starts with the current pointer position, and skips or takes characters from left to right for the specified number of columns. After execution, the pointer is positioned after the last character for the specified number of columns.

*This is the only clause in the **extract from** statement that allows you to specify a negative value.* When a negative number is specified, either of these clauses starts at one position to the left of the current pointer position and moves the pointer by the specified number of columns, from right to left. For the TAKE clause, characters are taken from left to right. After execution, the pointer is positioned at the leftmost column to which it was moved.

skip to COLUMN and take to COLUMN

Starts with the current pointer position and skips or takes characters up to but not including the specified column number. The designated column number must be in the range 1 to the column number of the final character of the source. After execution, the pointer is positioned at the specified column number.

Remarks: At least one TAKE clause must be specified. The only difference between the TAKE clause and the SKIP clause is that TAKE both moves the internal pointer and copies the processed characters into the variable, whereas SKIP simply moves the pointer.

Default Values: If EASEL OS/2 EE cannot find the specified characters, the pointer does not move, and, in TAKE clauses, the following values are copied to the specified variable:

Variable	Value
word	""
number	0
float	0.00
TEXT	""
COLUMN	""

Examples: response to line from Host
 extract from input
 take word NameX
 skip to 10
 take number IntX

Errors: The following are invalid clauses and will generate error messages:

skip [by] marker
take [by] marker
skip [by] last
take [by] last

See Also: **textual** Object Inquiry Built-in Function

fileid

fileid Definition

Purpose: To define a file identifier for use with File I/O Library sub-routines.

Model

fileid FILE_ID

Where: FILE_ID is an identifier for the name of a file.

Remarks: The file identifier is actually a file descriptor, or handle, that is used internally to identify the file. This provides a more efficient means of accessing the file.

Examples: fileid InFile
fileid OutFile

See Also: File I/O Library (Chapter 2)

find Action Statement

Purpose: To find a specified string of text in a textual region.

Model

find in TR_NAME [**from** [column] COL_NO [**line**] LINE_NO] STRING

Where: TR_NAME is the identifier for a textual region.
 COL_NO is an integer value representing a column number in the textual region.
 LINE_NO is an integer value representing a line number in the textual region.
 STRING is a string value that is the object of the **find**.

Use of ... **from** ...
Keywords: Specifies the column number to be used as the starting position of search. If the **from** clause is omitted, the search begins at the current text cursor position.

Remarks: The specified string is searched for from the current position of the text cursor, unless the **from** clause is specified. If the string is found, the text cursor moves to the first character in the first occurrence of the specified string, and the **found** built-in function is set to **true**. If the string is not found, the text cursor does not move, and the **found** built-in function is set to **false**.

If the cursor already lies on the first character of the string when the **find** statement is executed, the **found** built-in function is set to **true**, but the cursor does not move.

Examples: response to line "NEXT" from Host
 find in NewText "Page Two"

 response to Searcher
 find in Document SearchString # SearchString is a
 # string variable.

See Also: **found** Action Inquiry Built-in Function

font Reference

Purpose: To specify the default font for the program, the font used for a textual region, or the font used for any drawing statement.

Model

font { "system" | FONT_NAME }

Where: FONT_NAME is a string value representing a font name, or the identifier for a previously defined font. When specified for a textual region, it *must* be specified within the textual region definition.

Use of "system"
Keywords: Specifies that the PM system font should be used in a graphical drawing statement or in the program default font specification. The system font is a proportionally spaced font and therefore cannot be used in a textual region. The size of the system font is resolution-dependent, but it is not directly proportional to the size of the display.

Remarks: Fonts are either fixed-cell or resolution-dependent.

In a fixed-cell font, a text string occupies the same number of pixels regardless of the resolution of the monitor.

In a resolution-dependent font, a text string occupies the same proportion of any display. One font name identifies a family of fonts, one for each display resolution. EASEL OS/2 EE will use the appropriate font based on the type of display.

Resolution-dependent fonts are available for these graphics display adapters:

EGA	(640 x 350)
VGA	(640 x 480)
8514/A	(1024 x 768)

Only one font can be specified for each textual region.

Proportionally spaced, outline (scalable) PM fonts, and fonts defined with attributes (bold, italic, underline, or strikeout) cannot be used in textual regions.

Non-proportional PM fonts support the Extended ASCII Character Set for display of National Language characters in a textual region. EASEL OS/2 EE fonts do not support the Extended ASCII Character Set.

Any PM font can be displayed in a graphical region.

Using **screen size dialog units** defines a desktop coordinate system that is proportional to the size of the system font.

*EASEL
OS/2 EE
Fixed-Cell
Fonts*

Font Name	Total Size
asc37	4 x 9
asc59	7 x 11
asc79	9 x 12
asc510	5 x 10
asc713	7 x 13
asc88	8 x 8
asc814	8 x 14
caps814	8 x 14
asc1115	13 x 18
asc1521	17 x 24
medvt10	8 x 10
medvt12	8 x 12
medvt14	8 x 14
medvt16	8 x 16
medvt20	8 x 20
medvt24	8 x 24

*EASEL
OS/2 EE
Resolution-
Dependent
Fonts*

Screen Resolution	Font Name	Total Size
640x350	medium	8 x 8
	large	2 x 14
	mcaps	8 x 8
	lcaps	2 x 14
640x480	medium	8 x 12
	large	2 x 19
	mcaps	8 x 12
	lcaps	2 x 19
1024x768	medium	2 x 18
	large	9 x 30
	mcaps	2 x 18
	lcaps	9 x 30

mcaps and **lcaps** are uppercase fonts.

font

OS/2 Presentation Manager Fonts: See the font table on page 1-144.

Default Values: If no font is specified in a textual region definition, the default font (the font specified in the **font** environment declaration) will be used.

If no font is specified in an environment declaration, or if a proportionally spaced or outline (scalable) PM font was specified as the program default, the default font for the program is the **medium** resolution-dependent font.

Examples:

```
font "medium"      # Environment declaration

font "asc510"      # Environment declaration

textual region News
  window size 19 columns 5 lines
  at position 100 100
  font Courier_12_20 # Quotation marks not used around
                    # names of defined fonts

textual region News
  window size 19 columns 5 lines
  at position 100 100
  font "asc510"
```

See Also: **font is** Font Definition
text Drawing Statement

font is Font Definition

Purpose: To define a font for later use with the **font** Reference.

Model

```
font FONT_NAME is facename FACE_NAME
  size XSIZE YSIZE
  [bold] [italic] [underline] [strikeout] [outline]
```

Where: FONT_NAME is the identifier for a font.

FACE_NAME is a string value representing the style of the font.

XSIZE and YSIZE are the average sizes in pixels of the font. If the **outline** keyword is specified, XSIZE and YSIZE are the average sizes of the font in the coordinate system of the object that the text appears.

Use of **facename ...**
Keywords: Specifies the face name of a PM font or the string value representing an EASEL OS/2 EE fixed-cell or resolution-dependent font.

size ...
 Specifies the height and average width of a character in the font.

bold
 Makes the font bold.

italic
 Slants the font.

underline
 Puts a line on the baseline of each character.

strikeout
 Puts a line through the middle of each character.

outline
 Draws each character as an outline and then fills it. Outline fonts will scale. Outline text looks best large; it draws more slowly than non-scalable text. The **outline** keyword can be used only with PM fonts.

font is

<i>OS/2 Presentation Manager Fonts:</i>	Face Name	Type	Available Sizes
	Courier	fixed-width	8x7, 8x10, 8x13 9x8, 9x12, 9x16 10x10 12x10, 12x12, 12x15, 12x20 15x15
	Helv	proportional	5x6, 5x10, 5x13 6x8, 6x10, 6x12, 6x16 7x10, 7x12, 7x15, 7x20 8x12, 8x18, 8x24 9x15 10x18, 10x22 11x15, 11x29 13x22 14x19, 14x28, 14x37 28x28
	System Proportional	proportional	Depends on display resolution Specify a size of 0 0
	System Monospaced	fixed-width	8x8, 8x12, 8x16, 9x20
	Tms Rmn	proportional	4x6, 4x10, 4x13 5x12, 5x16 6x8, 6x10, 6x15, 6x19 7x10, 7x11, 7x12, 7x16, 7x21 9x15 10x14, 10x16, 10x20, 10x27 13x21, 13x26, 13x35 14x18 17x27

*EASEL
OS/2 EE
Fonts:* See the fixed-cell and resolution-dependent font tables on page 1-141.

Remarks: Once a FONT_NAME has been defined, it can be referenced wherever a font can be specified.

Proportionally spaced, outline (scalable) PM fonts, and fonts defined with attributes (bold, italic, underline, and strikeout) cannot be used in textual regions.

Non-proportional PM fonts support the Extended ASCII Character Set for display of National Language characters in a textual region.

Any PM font can be displayed in a graphical region.

Enter **font "system"** if you want to use the System Proportional font without an attribute (bold, italic, underline, strikeout, or outline) in a program. To use any other PM font, you must define it using the **font is** statement.

To use the System Proportional font in a **font is** statement, specify a size of 0 0. The appropriate size will be determined at runtime.

Examples:

```
font Newfont is facename "Helv"
    size 16 28 italic outline

font SystemItalic is
    facename "System Proportional"
    size 0 0
    italic
```

See Also:

- font** Reference
- multiline entry field** Definition
- text** Drawing Statement
- textual region** Definition

for each member loop Action Statement

Purpose: To specify a set of action statements to be performed on each member of a class.

Model

```
for each member of CLASS_NAME loop [LOOP_NAME]
```

```
  [ACTION_STATEMENT]...  
  [leave loop [LOOP_NAME] ]
```

```
end loop [LOOP_NAME]
```

Where: CLASS_NAME is the identifier for a previously defined class.

LOOP_NAME is the identifier for the loop.

ACTION_STATEMENT is an action statement.

*Use of
Keywords:*

leave loop

The **leave loop** action statement can be one of the action statements in a **for each member loop** statement. Upon execution, EASEL OS/2 EE will exit from the loop, and will continue execution at the statement following the **end loop** keywords. If you specify a **leave loop** statement, you must specify it within the definition of the loop itself. You cannot specify it in an action routine that is called from inside the loop.

leave loop LOOP_NAME

Specifies a name in the **leave loop** statement. This name might identify the loop in which the **leave loop** is specified, or it might identify an outer loop. If you specify an outer loop, EASEL OS/2 EE leaves the specified outer loop and all inner loops. If you do not specify a name in the **leave loop** statement, EASEL OS/2 EE exits from the innermost loop that contains the **leave loop** statement, even if the innermost loop has a name.

Remarks: Nesting of **for each member loop** statements is not allowed.

The **for each member loop** statement is used to loop through the members of a class. The built-in function **member** is used to access the name of the current class element. Class members can be accessed individually as well as by group.

When members are added to a class, they are added to the beginning of the class. Therefore, the order that objects will come out of a class is the reverse order in which they were added (last added to class will be the first member in the **for each member loop**).

If a **for each member loop** finishes normally, then the value of **member** is the empty string, which is also its initial value.

If a **leave loop** statement is used in the middle of a **for each member loop**, then the last assigned value to **member** will continue to be the value of **member**.

The **for each member loop** statement can be specified within a response definition, action routine definition, or EASEL OS/2 EE subroutine. It can be specified within a conditional action statement or within other loops, to any number of levels. You can specify blocks in loops, and vice versa. If you specify a block within a loop, execution of a **leave loop** statement causes all blocks contained in the loop to become inactive.

The identifier used to name the loop must be unique within the EASEL OS/2 EE program; there must be nothing else in the program with that name. Specifying a name in the **for each member loop** statement means that you *must* include a name in the **end loop** statement that exactly matches the name specified in the **for each member loop**.

Examples:

```
for each member of CLASS loop
  send member to errorlog
end loop
```

In the example above, the value of **member** the first time through is the last member added to the class CLASS.

See Also: **leave loop** Action Statement
member Special Inquiry Built-in Function

for each selected line loop

for each selected line loop Action Statement

Purpose: To perform a list of actions for each selected line in a multiple selection list box.

Model

```
for each selected line of LB_NAME loop [LOOP_NAME]

    [ACTION_STATEMENT]...
    [leave loop [LOOP_NAME] ]

end loop [LOOP_NAME]
```

Where: LB_NAME is the identifier for a list box.
LOOP_NAME is the identifier for the loop.
ACTION_STATEMENT is an action statement.

Use of **leave loop**
Keywords: The **leave loop** action statement can be one of the action statements in a **for each selected line loop** statement. Upon execution, EASEL OS/2 EE will exit from the loop, and will continue execution at the statement following the **end loop** keywords. If you specify a **leave loop** statement, you must specify it within the definition of the loop itself. You cannot specify it in an action routine that is called from inside the loop.

leave loop LOOP_NAME

Specifies a name in the **leave loop** statement. This name might identify the loop in which the **leave loop** is specified, or it might identify an outer loop. If you specify an outer loop, EASEL OS/2 EE leaves the specified outer loop and all inner loops. If you do not specify a name in the **leave loop** statement, EASEL OS/2 EE exits from the innermost loop that contains the **leave loop** statement, even if the innermost loop has a name.

Remarks: The actions in the loop are repeated for each selected line in the multiple selection list box. The built-in function **selected line** is used to reference the currently selected line within the **for each selected line loop**. Be careful not to use the **selected line from** function, since it will not change each time through the loop as **selected line** does.

The **for each selected line loop** statement can be specified within a response definition, action routine definition, or EASEL OS/2 EE subroutine. It can be specified within a conditional action statement or within other loops, to any number of levels. You can specify blocks in loops, and vice versa. If you specify a block within a loop, execution of a **leave loop** statement causes all blocks contained in the loop to become inactive.

The identifier used to name the loop must be unique within the EASEL OS/2 EE program; there must be nothing else in the program with that name. Specifying a name in the **for each selected line loop** statement means that you *must* include a name in the **end loop** statement that exactly matches the name specified in the **for each selected line loop**.

Do not use a **read** or **insert** statement to change the contents of a list box while executing this loop. You can use a **remove** statement to remove one or more lines.

Examples:

```

response to item CreateQuery
  call ProcessQueryDialogBox(DidOKCUA)
  if (DidOKCUA) then
    copy "report on " to NameList
    for each selected line of QueryList loop
      copy NameList " " (textual line (selected line)
        from QueryList) to NameList
    end loop
    send NameList "\n" to LocalApp1
  end if

```

See Also: **leave loop** Action Statement
selected line Special Inquiry Built-in Function

for loop Action Statement

Purpose: To specify a set of action statements to be performed repeatedly, until the integer variable reaches a specified value.

Model

```
for INTEGER_VAR_NAME = VALUE1 to VALUE2 [by VALUE3] loop [LOOP_NAME]

    [ACTION_STATEMENT]...
    [leave loop [LOOP_NAME] ]

end loop [LOOP_NAME]
```

Where: INTEGER_VAR_NAME is a previously defined integer variable.

VALUE1, VALUE2, and VALUE3 are integer values.

LOOP_NAME is the identifier for the loop.

ACTION_STATEMENT is an action statement.

*Use of
Keywords:* **leave loop**

The **leave loop** action statement can be one of the action statements in a **for loop** statement. Upon execution, EASEL OS/2 EE will exit from the loop, and will continue execution at the statement following the **end loop** keywords. If you specify a **leave loop** statement, you must specify it within the definition of the loop itself. You cannot specify it in an action routine that is called from inside the loop.

leave loop LOOP_NAME

Specifies a name in the **leave loop** statement. This name might identify the loop in which the **leave loop** is specified, or it might identify an outer loop. If you specify an outer loop, EASEL OS/2 EE leaves the specified outer loop and all inner loops. If you do not specify a name in the **leave loop** statement, EASEL OS/2 EE exits from the innermost loop that contains the **leave loop** statement, even if the innermost loop has a name.

Remarks: The **for loop** statement extends **while loop** support by allowing a specification of an initial value for the loop variable. It also allows automatic incrementing or decrementing of the loop variable.

When a **for** statement is encountered, INTEGER_VAR_NAME (IVN) is initialized to VALUE1. If IVN is less than VALUE2, the loop is executed. The next time through the loop, IVN is incremented by VALUE3. (If VALUE3 is not provided, then a value of 1 will be used. A negative value can be used for VALUE3 to cause VALUE1 to go from a larger to a smaller value.) Then IVN is tested against VALUE2. The value of IVN when the loop is ended will be the first increment value that caused the loop to fail.

If VALUE3 is zero, the loop will not be executed. If the current value of IVN is less than VALUE2 and VALUE3 is negative, the loop is terminated. Similarly, if the current value of IVN is greater than VALUE2 and VALUE3 is positive, the loop is terminated.

The **for loop** statement can be specified within a response definition, an action routine definition, or an EASEL OS/2 EE subroutine. It can be specified within a conditional action statement or within other loops, to any number of levels. You can specify blocks in loops, and vice versa. If you specify a block within a loop, execution of a **leave loop** statement causes all blocks contained in the loop to become inactive.

The identifier used to name the loop must be unique within the EASEL OS/2 EE program; there must be nothing else in the program with that name. Specifying a name in the **for loop** statement means that you *must* include a name in the **end loop** statement that exactly matches the name specified in the **for loop**.

Examples:

```
integer A
for A = 1 to 5 loop           #uses default increment of 1
    send A to errorlog
end loop
```

The loop can also go down, in which case VALUE3 should be negative.

```
integer A
for A = 5 to 1 by -2 loop
    send A to errorlog
end loop
```

See Also: **leave loop** Action Statement

foreground at and **background at** Object Inquiry Built-in Functions

Purpose: To find the foreground or background colors of a specified column and line position in a colored textual region.

Model

{ **foreground** | **background** } **at** [column] COL_NO [line] LINE_NO **char**
of TR_NAME

Where: COL_NO is an integer value representing a column in the textual region.

LINE_NO is an integer value representing a line in the textual region.

TR_NAME is a string value representing the identifier for the textual region.

Remarks: **foreground at** and **background at** each return an integer value associated with the color of the foreground or background of the character at the specified line and column.

When these functions are used with colored textual regions, the values returned represent the colors of the individual characters in the colored textual region. If these functions are used with uncolored textual regions, the values returned are the same as the **foreground of** and **background of** functions, representing the colors specified in the textual region definition.

Examples:

```
# If the text at the position selected does not
# have a red (color 1) background, then make it red
response to TR_7
  if (background at column xcoord line ycoord char
    of TR_7 != 1) then
    make TR_7 column xcoord line ycoord background red
  end if
```

foreground at and **background at**

```
# If the text at the position selected does not
# have a blue (color 4) foreground, then make it blue
response to TR_8
  if (foreground at column xcoord line ycoord char
      of TR_8 != 4) then
    make TR_8 column xcoord line ycoord foreground blue
  end if
```

See Also: **background of** Object Inquiry Built-in Function
color Keyword
foreground of Object Inquiry Built-in Function

foreground of Object Inquiry Built-in Function

Purpose: To find and use the foreground color of a specified object.

Model

foreground of OBJECT_NAME

Where: OBJECT_NAME is a string value representing the identifier for an object.

Remarks: For graphical, image, and textual regions, the value returned is the integer value of the color associated with the foreground of the specified region.

For keys, the value returned is the integer value of the color associated with the object. If the object is invisible, a negative value will be returned.

Default Values: If no color is specified for the object, the **foreground of** function will return the integer **7** (white). This is also the case if the object is invisible (and no color has been specified).

Examples: response to Labels
make Labels foreground color foreground of Bulletin

See Also: **background of** Object Inquiry Built-in Function
color Keyword
foreground at and **background at** Object Inquiry Built-in Functions
make COLOR Action Statement

found Action Inquiry Built-in Function

Purpose: To determine whether the **find** action statement was successful in finding a string in a specified textual region.

Model

found

Remarks: **found** is a boolean function, returning the value **true** if the string was found, and **false** if it was not.

The **found** function can be used *only* to test the success of a **find** action statement.

**Default
Values:** **false**

Examples:

```
response to More
  find in Document SearchString
  while (found) loop
  :
  end loop

response to NewPage
  find in Document "page #"
  if (found) then
  :
  end if
```

See Also: **find** Action Statement

Frame Attributes

Remarks: Frame attributes are standard Presentation Manager (PM) user interface components that allow the user to interact with the window. PM controls the user interaction with frame attributes independently of EASEL OS/2 EE.

Frame attributes can be specified in all EASEL OS/2 EE region types except sense regions. Regions can have any combination of the following frame attributes:

title bar

maximize button

minimize button

horizontal scroll bar

vertical scroll bar

size border or **border** (standard border)

system menu

action bar

The **border** (standard border) specification will cause a one-pixel wide border in the PM border color to be displayed only if another frame attribute is specified for the region.

freesize Special Inquiry Built-in Function

Purpose: To find the byte size of the single largest available block of memory free to be used by the EASEL OS/2 EE program. This function can be used to monitor memory usage within your program.

Model

freesize

Remarks: The **freesize** function is used in conjunction with the **response to low memory** response statement and the **set low memory threshold** and **squeeze memory** action statements.

The **freesize** function returns an integer. The value includes the sum of all available memory only when it is checked *immediately after* the **squeeze memory** action statement.

Since EASEL OS/2 EE allocates memory in 64K chunks, the largest value **freesize** can be is 64K.

The **squeeze memory** statement is often used in conjunction with **freesize**, since it will reorder memory, collecting all free memory together.

Examples:

```

response to low memory
  squeeze memory
  if (freesize < 200) then
    change to program Alternate_Choices
  end if

```

See Also: **response to low memory** Response Definition
set low memory threshold Action Statement
squeeze memory Action Statement

function Declaration

Purpose: To declare an external function.

Model

```
function FUNCT_NAME( [TYPE1:ARG_NAME[,  
                      TYPE2:ARG2_NAME]... ] )  
  returns TYPE  
  library LIB_NAME
```

Where: FUNCT_NAME is the identifier for the function.
TYPE_n is: **integer**, **string**, **boolean**, or **float**.
ARG_n_NAME is a variable of type TYPE_n.
TYPE is the return type of the function (integer, string, boolean, or float).
LIB_NAME is the identifier for an OS/2 dynamic link library.

Remarks: An external function must be declared before it is called (invoked).

Libraries provided with EASEL OS/2 EE have include files that contain declarations for all the external functions in the library.

A function is invoked by referencing the function name, followed by its arguments enclosed in parentheses, whenever a value of the same type as the return type can be used.

Examples: function SIN(float:X)
 returns float
 library "mathlib"

Errors: If an external function is called but never declared, an error is generated at compile time.

See Also: **subroutine** Declaration
 Chapter 2, Functions and Subroutines
 Chapter 4, Include Files

graphical region Definition

Purpose: To define a graphical region, which is a rectangular portion of the screen in which other objects can be drawn.

Model

```
[ enabled | disabled ] [ visible | invisible ] [ [ color ] COLOR ]
[primary] [graphical] region GR_NAME
    size XSIZE YSIZE
    at { [position] X Y | default position }
    [in PARENT_NAME1 [in PARENT_NAME2]... ]
    [window size WINDOW_XSIZE WINDOW_YSIZE
      at [position] WINDOW_X WINDOW_Y]
    [ [color] COLOR foreground]
    [priority is PRIORITY_NO]
    [parameter is PAR_STRING ]
    [ [ size | [color] COLOR ] border]
    [title bar TITLE_STRING]
    [system menu]
    [horizontal scroll bar [scroll by SCROLL_INC] ]
    [vertical scroll bar [scroll by SCROLL_INC] ]
    [no scale]
    [action bar AB_NAME]
    [minimize button [using MINIMIZE_ICON_FILE] ]
    [maximize button]
    [pointer [is] POINTER_FILE]

    [DRAWING_STATEMENT]...
```

Where: COLOR is a string or integer value representing a color.

GR_NAME is the identifier for the graphical region.

XSIZE and YSIZE are integer values representing the dimensions of the region's viewport in the parent's coordinate system. These refer to the dimensions of the viewport itself, and do not include the space occupied by the frame.

X and Y are integer values representing the position of the viewport in the parent's coordinate system (not the position of the outside frame).

PARENT_NAME1 and PARENT_NAME2 are identifiers for parent objects. If it is a primary region, you cannot specify ancestry; it is always **desktop**. To place a region outside of a primary region, specify its ancestry as **in desktop**

WINDOW_XSIZE and WINDOW_YSIZE are integer values representing the dimensions of the region's window. The window defines what portion of the region's contents are visible in the viewport.

WINDOW_X and WINDOW_Y are integer values representing the position of the region's window.

PRIORITY_NO is an integer value representing the priority of the region. The primary region and regions whose ancestry is desktop cannot be assigned a priority.

PAR_STRING is a string value representing the region's parameter.

TITLE_STRING is a string value representing the title that will appear in the region's title bar.

SCROLL_INC is an integer value representing the number of units in the region's coordinate system by which the region's window position will be changed when one of the arrows in the scroll bar is pressed. When paging, the page increment is the X or Y window size minus one scroll increment.

AB_NAME is a string value representing the identifier for an action bar template.

MINIMIZE_ICON_FILE is a string value representing the name of the file, including the extension, containing the icon to use when the region is minimized.

POINTER_FILE is a string value representing the name of the file containing the bit map to use as a pointer whenever the pointer is over this region.

DRAWING_STATEMENT is a drawing statement.

*Use of
Keywords:*

primary

Specifies that this region is the program's primary region.

size ...

Defines the size of the region's viewport, and by default the size of its window, in X Y coordinates of the parent. The default window size can be overridden by the **window size** attribute.

window size ...

Defines the size of the region's window, in X Y coordinates of the object.

... border

Specifies the type of border present around the region. If **border** or **COLOR border** is specified, a one-pixel border of the color indicated (white, by default) will be drawn. If **size border** is specified, the region will be surrounded by a sizing border that can be used to resize the region. If other frame attributes are specified for the region, then the **border** specification will cause a one pixel wide border in the PM border color to be displayed.

title bar ...

Specifies that the region will contain a title bar. This will generate a title bar that can be used for repositioning the region. The title bar will contain the text specified.

system menu

Specifies that the region will contain a system menu. The system menu button will be in the upper left corner.

horizontal scroll bar ...

Specifies that the region will contain a horizontal scroll bar. The scroll bar will be located along the bottom edge of the region just inside the border, if one exists. The region's contents will be scrolled as specified by `SCROLL_INC`.

vertical scroll bar ...

Specifies that the region will contain a vertical scroll bar. The scroll bar will be located along the far right-hand edge of the region just inside the border, if one exists. The region's contents will be scrolled as specified by `SCROLL_INC`.

no scale

Specifies that scaling of the region's contents should not occur when changing the region size (i.e., the window size is constrained by the viewport). When the size of the region is changed, its window size is changed in parallel. Resizing the region thus exposes or covers more of the region contents, rather than causing the contents to be rescaled to fit into the new size of the region. Note that the **window size** attribute will have no effect if this attribute is specified.

action bar ...

Specifies that the region will contain an action bar.

minimize button ...

Specifies that the region will contain a minimize button in the upper right corner.

graphical region

maximize button

Specifies that the region will contain a maximize button in the upper right corner. (The maximize button will turn into a restore button when the region is maximized.)

pointer is ...

Specifies the file containing the bit map to use as a pointer.

Remarks: In a windowed application, the primary region must be the first object you define. There can be only one primary region. All other regions and keys are, by default, children of the primary region.

Only the **region** specification, the identifier, the **at** [**position**] specification, and the **size** specifications are required, and they must be specified in the order shown in the model. Other attributes are optional, using the defaults shown below.

Any of the attributes after ancestry can be specified in any order.

When you define a region's size, you are actually defining two characteristics: the region's viewport and the region's window. You can also specify the window size separately.

The region size only includes the area of the viewport itself. It does not include the frame attributes, which are external to the viewport. If the size of the primary region is the same as the screen size, you will not be able to see any frame attributes.

Double-clicking on the system menu or selecting the "Close" choice from the system menu destroys the region, if there is no **on close** response to invoke for the region.

In a windowed application, disabling a region disables its viewport and its frame, and also disables the region's children.

In a fullscreen application, a region is disabled by default, and a region's selectability does not affect the selectability of its children.

<i>Default</i>	SELECTABILITY:	enabled
<i>Values:</i>	VISIBILITY:	visible
	COLOR (background):	black
	ANCESTRY (primary):	in desktop
	ANCESTRY:	in primary region
	COLOR (foreground):	white
	PRIORITY:	0
	BORDER:	invisible
	PARAMETER:	""
	COLOR (border):	white
	SCROLL_INC:	one unit

Examples: enabled visible color 26
primary graphical region PrimaryWindow
size 510 280
at position 26 40
color 27 foreground
size border
title bar "Primary Window"
system menu
horizontal scroll bar scroll by 6
vertical scroll bar scroll by 16
no scale
action bar PrimaryWindowActionBar
minimize button using "minbuttn.ico"
maximize button
pointer is "myptr.ptr"
move to 255 260
font "large" black text center "ACME COMPANY"
move -1 1
font "large" black text center "ACME COMPANY"
move -1 1
font "large" blue text center "ACME COMPANY"

In the example above, color 26 picks up the PM window background color, and color 27 picks up the PM window foreground color.

<i>See Also:</i>	border Attribute Definition	parameter is Attribute Definition
	color Keyword	
	enabled and disabled Attribute Definitions	priority is Attribute Definition
	in Keyword	visible and invisible Attribute Definitions

group box Definition

Purpose: To define a group box.

Model

```
[ enabled | disabled ] [ visible | invisible ]  
group box GB_NAME  
  size WIDTH HEIGHT  
  at [position] X Y  
  in DB_NAME  
  [parameter is PAR_STRING]  
  [text TEXT_STRING]
```

Where: GB_NAME is the identifier for a group box.

WIDTH and HEIGHT are integer values representing the dimensions of the group box in dialog units.

X and Y are integer values representing the position of the group box in dialog units.

DB_NAME is the identifier for the dialog box or dialog region parent object.

PAR_STRING is a string value representing the parameter of the group box.

TEXT_STRING is a string value representing the text that will be displayed in the upper left-hand corner of the group box.

Use of **text...**
Keywords Specifies the text that will be displayed by the group box.

Remarks: A group box is a dialog control, and it must have a dialog box or dialog region as its parent.

Only the **group box** keywords, the identifier, the **at** [position] specification, the **in** DB_NAME, and the **size** specification are required in the definition.

When specifying attributes, **parameter** and **text** can be specified in any order; all other attributes must be specified in the order shown in the model.

Enabling or disabling the group has no effect.

Default **SELECTABILITY:** **enabled**
Values: **VISIBILITY:** **visible**
 PARAMETER: **""**
 TEXT: **""**

Examples: invisible dialog box SaveFile
 size 160 100 at position 200 200
 dialog border
 title bar "Save Output File"
 system menu

 group box Destination
 size 100 40
 at position 6 20 in SaveFile
 text "Destination"

 checked radio button HardDisk
 size 48 11
 at position 10 30 in SaveFile
 group is Options
 text "to Hard Disk"

 radio button Network
 size 48 11
 at position 60 30 in SaveFile
 group is Options
 text "to Network"

handle of Object Inquiry Built-in Function

Purpose: To retrieve the PM window handle associated with an object.

Model

handle of OBJECT_NAME

Where: OBJECT_NAME is a string value representing the identifier for an object. The object cannot be a key or a sense region.

Remarks: The function returns an integer value that is the PM handle (or identifier) for the object. This value can be passed to external functions or subroutines that can use the value.

Note that the consequences of the use of this value by an external function or subroutine are the responsibility of the developer. EASEL OS/2 EE does not guarantee proper operation when this handle is used by external code. (For instance, the code must not delete the window.)

Examples:

```
subroutine MySubr(integer: Handle)
    library "mylib"
    :
integer PrimaryHandle
    :
copy (handle of Primary) to PrimaryHandle
call MySubr(PrimaryHandle)
```

highlight Action Statement

Purpose: To turn highlighting on or off. When highlighting is on, a correct selection of an object causes a frame to be displayed around the object for a moment.

Model

highlight { [**color**] COLOR | **off** }

Where: COLOR is a string or integer value representing an EASEL OS/2 EE color.

Remarks: The **highlight** statement is typically specified in a **response to start** response definition.

The highlight frame is always drawn in *complement mode*—the inverse of the background color—to ensure that the frame will always be visible. The specified color of the frame will be displayed in that color when the frame is against a black background, but when it is moved to another color background, it will change color as necessary.

White is the most useful color for the highlight in windowed applications: when inverted, it will always be more visible than any other specified color. For example, it will be white against a black background, black against a white background, green against a magenta background, and so on. Note that a black frame is not useful, since inverted black is not visible against *any* background color.

Color 25 is the most useful color for the highlights in fullscreen applications.

You can change the amount of time the highlight appears by setting the value of the **HLDELAY** command in the CONFIG.ESL file. Initially, the value of **HLDELAY** is 8000; a value between 0 and 32000 can be specified. The larger the value, the longer EASEL OS/2 EE will flash the highlight. The default value is 0; thus, if **HLDELAY** is specified without a value, the highlight will be drawn and then immediately erased.

Default Values: **off**

highlight

Examples: response to start
 highlight white

 response to Undo
 highlight off

See Also: **CONFIG.ESL** File (Chapter 3)

Identifiers

Purpose: To identify an entity for later reference.

Model

Any sequence of characters that begins with an uppercase letter (A-Z). You must substitute your own identifiers for UPPERCASE words shown in the syntax models in this publication.

Remarks: An identifier, or name, can contain any number of characters that fit on one line. After the initial uppercase letter, the remaining characters in the name can be uppercase or lowercase letters, digits, and/or underscore characters.

Note: The case in which you specify an identifier is significant. The identifier RedKey is not the same as the identifier Redkey.

When you reference the name of an object that has ancestry, you must specify as many ancestors as necessary to identify the name uniquely. You can use the shorter form as a name only if no other object has the same name.

The EASEL OS/2 EE entities that require identifiers are action routines, local and remote applications, classes, constants, objects, action bars, pulldowns, button, choices, patterns, and variables. The EASEL OS/2 EE entities that can have optional identifiers are blocks, loops, and separators.

Examples: key Stop at position 100 100 # identifier is Stop
integer variable X2 # identifier is X2

if Conditional Action Statement

Purpose: To specify a set of action statements to be performed if a specified condition is true, and other alternative optional action statements to be performed if the condition is false.

Model

```
if ( BOOLEAN_EXPRESSION ) then  
    [ACTION_STATEMENT] ...  
[else  
    [ACTION_STATEMENT] ... ]  
end if
```

Where: BOOLEAN_EXPRESSION is a boolean value.
ACTION_STATEMENT is an action statement.

Remarks: The **if** statement is a conditional action statement. It can be specified within a response definition or an action routine definition. It can be specified within a **loop** statement, or even within another conditional action statement. Blocks can be specified within conditional action statements, and vice versa.

You specify both the condition and the actions in a conditional action statement. A conditional action statement must begin with the **if** statement; it must contain a **then** clause; and it must end with the **end if** keywords. The **else** clause is optional.

EASEL OS/2 EE first evaluates the boolean expression specified in the **if** statement. If the boolean expression is **true**, EASEL OS/2 EE performs the action statements specified in the **then** clause; the **else** clause (if specified) is ignored. If the boolean expression is **false**, EASEL OS/2 EE performs the action statements specified in the **else** clause (if specified); the **then** clause is ignored.

After executing the **then** clause or the **else** clause, EASEL OS/2 EE continues normal program execution at the action statement (if any) immediately following the **end if** keywords.

You can nest conditional action statements to any number of levels. Remember that you must specify the **end if** keywords corresponding to each **if** keyword.

Examples:

```

response to Tester2
    if (TestSuccessFlag) then
        if (A > B) then
            :
        else
            :
        end if
    end if

action CheckSize is
    if (A > B and Count < MaxCount) then
        append Count to Log
        send Count to LogProg
    end if

```

See Also: Expressions, Operators, and Operands
while loop Action Statement

image region Definition

Purpose: To define an image region, which is a rectangular portion of the screen in which bitmap images can be displayed.

Model

```
[ enabled | disabled ] [ visible | invisible ] [ [color] COLOR ]
[primary] image region IR_NAME
    size XSIZE YSIZE
    at { [position] X Y | default position }
    [in PARENT_NAME1 [in PARENT_NAME2]... ]
    [window size WINDOW_XSIZE WINDOW_YSIZE
      at [position] WINDOW_X WINDOW_Y]
    [ [color] COLOR foreground]
    [priority is PRIORITY_NO]
    [parameter is PAR_STRING]
    [ [ size | [color] COLOR ] border]
    [title bar TITLE_STRING]
    [system menu]
    [horizontal scroll bar [scroll by SCROLL_INC] ]
    [vertical scroll bar [scroll by SCROLL_INC] ]
    [ no scale | sample | preserve foreground | preserve background ]
    [ no cache | cache image | cache viewport | cache CACHE_SIZE bytes ]
    [action bar AB_NAME]
    [minimize button [using MINIMIZE_ICON_FILE] ]
    [maximize button]
    [pointer [is] POINTER_FILE]
    [file FILE_NAME [using IMAGE_MAP] ]
```

Where: COLOR is a string or integer value representing a color.

IR_NAME is the identifier for the image region.

XSIZE and YSIZE are integer values representing the dimensions of the region's viewport in the parent's coordinate system. These refer to the dimensions of the viewport itself, and do not include the space occupied by the frame.

X and Y are integer values representing the position of the viewport in the parent's coordinate system (not the position of the outside frame).

PARENT_NAME1 and PARENT_NAME2 are identifiers for parent objects. If it is a primary region, you cannot specify the ancestry; it is always **desktop**. To place a region outside of a primary region, specify its ancestry as **in desktop**.

WINDOW_XSIZE and WINDOW_YSIZE are integer values representing the dimensions of the region's window. The window defines what portion of the region's contents are visible in the viewport.

WINDOW_X and WINDOW_Y are integer values representing the position of the region's window.

PRIORITY_NO is an integer value representing the priority of the region. The primary region and regions whose ancestry is desktop cannot be assigned a priority.

PAR_STRING is a string value representing the region's parameter.

TITLE_STRING is a string value representing the title that will appear in the region's title bar.

SCROLL_INC is an integer value representing the number of units in the region's coordinate system by which the region's window position will be changed when one of the arrows in the scroll bar is pressed. When paging, the page increment is the X or Y window size minus one scroll increment.

CACHE_SIZE is an integer value representing the maximum allowable cache size.

AB_NAME is a string value representing the identifier for an action bar template.

MINIMIZE_ICON_FILE is a string value representing the name of the file, including the extension, containing the icon to use when the region is minimized.

POINTER_FILE is a string value representing the name of the file, including the extension, containing the bit map to use as a pointer whenever the pointer is over this region.

FILE_NAME is a string value representing the name of the file, including the extension, used for the image region's initial contents.

IMAGE_MAP is a string value representing the identifier for a previously defined image map, which maps the colors of an image.

image region

*Use of
Keywords:*

primary

Specifies that this region is the program's primary region.

size ...

Defines the size of the region's viewport, and by default the size of its window, in X Y coordinates of the parent. The default window size can be overridden by the **window size** attribute.

window size ...

Defines the size of the region's window, in X Y coordinates of the object.

... border

Specifies the type of border present around the region. If **border** or **COLOR border** is specified, a one-pixel border of the color indicated (white, by default) will be drawn. If **size border** is specified, the region will be surrounded by a sizing border that can be used to resize the region. If other frame attributes are specified for the region, then the **border** specification will cause a one pixel wide border in the PM border color to be displayed.

title bar ...

Specifies that the region will contain a title bar. This will generate a title bar that can be used for repositioning the region. The title bar will contain the text specified.

system menu

Specifies that the region will contain a system menu. The system menu button will be in the upper left corner.

horizontal scroll bar ...

Specifies that the region will contain a horizontal scroll bar. The scroll bar will be located along the bottom edge of the region just inside the border, if one exists. The region's contents will be scrolled as specified by **SCROLL_INC**.

vertical scroll bar ...

Specifies that the region will contain a vertical scroll bar. The scroll bar will be located along the far right-hand edge of the region just inside the border, if one exists. The region's contents will be scrolled as specified by **SCROLL_INC**.

no scale

Specifies that scaling of the region's contents should not occur when changing the region size (i.e., the window size is constrained by the viewport). When the size of the region is changed, its window size is changed in parallel. Resizing the region thus exposes or covers more of the region contents, rather than causing the contents to be rescaled to fit into the new size of the region. Note that the **window size** attribute will have no effect if this attribute is specified.

sample

Specifies that scaling will be done using the sample method, which arbitrarily selects the scanlines to be retained, and discards the others without examining them.

preserve foreground

Specifies that scaling will be done giving preference to preserving the foreground color.

preserve background

Specifies that scaling will be done giving preference to preserving the background color.

no cache

Specifies that the system will decide what cache type to choose based on analysis of currently available memory.

cache image

Specifies that the entire image will be stored in RAM.

cache viewport

Specifies that the portion of the image in the image region's viewport will be stored in RAM.

cache ... bytes

Specifies that the portion of the image that `CACHE_SIZE` bytes will hold will be stored in RAM.

action bar ...

Specifies that the region will contain an action bar.

minimize button ...

Specifies that the region will contain a minimize button in the upper right corner.

image region

maximize button

Specifies that the region will contain a maximize button in the upper right corner. (The maximize button will turn into a restore button when the region is maximized.)

pointer is ...

Specifies the file containing the bit map to use as a pointer.

file ...

Specifies the file used for the image region's initial contents.

Remarks: In a windowed application, the primary region must be the first object you define. There can be only one primary region. All other regions and keys are, by default, children of the primary region.

Only the **region** specification, the identifier, the **at** [**position**] specification, and the **size** specifications are required, and they must be specified in the order shown in the model. Other attributes are optional, using the defaults shown below.

Except for the **file** option, which must be specified last, any of the attributes after ancestry can be specified in any order.

When you define a region's size, you are actually defining two characteristics: the region's viewport and the region's window. You can also specify the window size separately.

The region size only includes the area of the viewport itself. It does not include the frame attributes, which are external to the viewport. If the size of the primary region is the same as the screen size, you will not be able to see any frame attributes.

Double-clicking on the system menu or selecting the "Close" choice from the system destroys the region, if there is no **on close** response to invoke for the region.

In a windowed application, disabling a region disables its viewport and its frame, and also disables the region's children.

In a fullscreen application, a region is disabled by default, and a region's selectability does not affect the selectability of its children.

<i>Default</i>	SELECTABILITY	enabled
<i>Values:</i>	VISIBILITY:	visible
	COLOR (background):	black
	ANCESTRY (primary):	in desktop
	ANCESTRY:	in primary region
	COLOR (foreground):	white
	PRIORITY:	0
	BORDER:	invisible
	PARAMETER:	""
	COLOR (border):	white
	SCROLL_INC:	one unit
	SCALE:	sample
	CACHE:	cache image

Examples: enabled visible color 25 image region IC
size 465 275
at 90 20
color 8 foreground
size border
title bar "Image Region"
system menu
horizontal scroll bar scroll by 20
vertical scroll bar scroll by 20
no scale
no cache
minimize button using "ic2.ico"
maximize button
pointer is "SOLDER.PTR"
file "circuit2.pcx"

<i>See Also:</i>	border Attribute Definition	in Keyword
	color Keyword	parameter is Attribute Definition
	enabled and disabled	
	Attribute Definitions	priority is Attribute Definition
	imagemap Definition	visible and invisible Attribute Definitions

imagemap Definition

Purpose: To map the colors of an image region.

Model

imagemap IMAGE_MAP is
OLD_COLOR NEW_COLOR[, OLD_COLOR NEW_COLOR]...

Where: IMAGE_MAP is a string value representing the identifier for the image map.

OLD_COLOR is a color keyword or number that specifies a color to map. The actual value can be an integer variable, integer constant, integer literal, or color keyword.

NEW_COLOR is a color keyword or number that specifies a new color for the old color specified. The actual value can be an integer variable, integer constant, integer literal, or color keyword.

Remarks: The image map tells EASEL OS/2 EE how to color an image. Different software produces different color schemes, so that a red object in the original image might be displayed as blue in EASEL OS/2 EE. EASEL OS/2 EE supports 16 colors, and because some software can support more, undesirable results may occur as different colors in the original are mapped to the same color in EASEL OS/2 EE.

Once the image map has been defined, you can specify it in an image region definition with the **file** option, or in a **read** action statement.

Examples:

imagemap Colormap1 is	# You can use:
black red,	# keywords
White Magenta,	# constants
Red Yellow,	# variables
4 1	# literals

See Also: **image region** Definition
read Action Statement

in Keyword

Purpose: To specify ancestry for an object.

Model

in PARENT_NAME

- Where:* PARENT_NAME is the identifier for the object’s parent and can itself specify ancestry.
- Remarks:*

The ancestry specification must appear within the object definition or reference.

In a windowed application, the PARENT_NAME can be **desktop**, which refers to the PM desktop. In a fullscreen application, the PARENT_NAME can be **master**, which refers to the EASEL OS/2 EE background screen.

The ancestry attribute determines whether the object is to be positioned within the coordinate system of another object, of the PM desktop, or of the EASEL OS/2 EE screen. If ancestry is specified, that object is called the *child* of the *parent* object.

For a description of valid parent types associated with each object, see “Object Types and Ancestry” on page 1-239.

A parent object can have any number of children as long as the parent is a valid parent type of the child object. Each child object can have only one direct parent.

If the parent object is disabled, its children are also implicitly disabled (in windowed applications only). If the parent is made invisible, its children are also implicitly made invisible. If the parent is deleted or cleared, its children are deleted, unless the **clear graphics** statement is used. Note that using the **change** statement also deletes the children, unless the **change graphics** statement is used.

Since two objects might have the same name, whenever you reference a child object, you must specify a name containing enough of the object’s ancestry to distinguish it from other objects.

The primary window has the desktop as its parent. (This is the default and cannot be specified.)

in

A dialog box has the desktop as its parent. (This is the default and cannot be specified.)

Examples: disabled key Meter at position 50 50 in Panel in Control

textual region Panel window size 10 columns 5 lines
at position 100 100 in Control

make Panel in window visible

<i>See Also:</i>	check box Definition	group box Definition
	combination box	image region Definition
	Definition	key Definition
	dialog box Definition	list box Definition
	dialog region Definition	multiline entry field
	dropdown list Definition	Definition
	extract from Action	Object Types and Ancestry
	Statement	radio button Definition
	graphical region	static text Definition
	Definition	sense region Definition
		push button Definition
		textual region Definition

include Reference

Purpose: To insert an ASCII text file, which contains EASEL OS/2 EE source code, into the current EASEL OS/2 EE program. This feature is especially useful when you want to use the same sequence of statements in more than one program.

Model

include FILE_NAME

Where: FILE_NAME is a string literal representing the name of a file, including the extension, that contains EASEL OS/2 EE statements.

Remarks: You can specify any number of **include** action statements in an EASEL OS/2 EE program.

The file is inserted at the time that the program is compiled. EASEL OS/2 EE inserts all of the statements contained in the included file at the location at which the **include** statement is specified. After processing the last line of the inserted file, EASEL OS/2 EE continues to process the original file at the line following the **include** statement line.

The string literal that you specify in the **include** statement must be an OS/2 file name. Although the examples shown in this section use the extension .INC (for INclude), you can specify any other extension or none at all.

Nested include Included files can be nested; that is, each file can contain **include** statements. You can nest **include** statements to 20 levels. Make sure that no file includes itself or any other file that has already been included. If it does, each of the two files will cause the other file to become part of it each time EASEL OS/2 EE encounters the **include** statement, and inclusion will never end.

Files:

Examples:

```

response to Grapher
  move to xcenter ycenter
  include Points

action Logo is
  add disabled key Logo at position 100 100
  include "graphics.inc"
```

include

Errors: EASEL OS/2 EE generates error messages if you specify an invalid file name, too many levels of nesting, or a file that cannot be accessed or opened.

input Response Inquiry Built-in Function

Purpose: To find and use the input data that stimulated the current response.

Model

input

Remarks: The **input** function should be specified only in a **response to char** or **response to line** response definition.

The input data can be received from the keyboard or from an application.

Examples: response to char from keyboard
append input to AccumulatedLine

Errors: If used elsewhere than in a **response to char** or **response to line** response statement, input will return the null string.

See Also: **response to char** and **response to line** Response Definitions

insert Drawing Statement

Purpose: To insert text at a specified location in a textual region or list box.

Model #1 (for Textual Regions):

```
insert { [ at cursor | after cursor ] { TEXT_STRING |  
        [ at cursor | after cursor ]  
        segment [column] COL [line] LN  
          [thru [column] COL [line] LN] from TR_NAME |  
        block [column] COL [line] LN  
          [thru [column] COL [line] LN] from TR_NAME |  
        [colored] file FILE_NAME }
```

Model #2 (for List Boxes):

```
insert { TEXT_STRING | file ASCII_FILE_NAME }
```

Where: TEXT_STRING is a string value representing the text to be inserted.

COL is an integer value representing the beginning and ending column numbers in the block or segment.

LN is an integer value representing the beginning and ending line numbers in the block or segment.

TR_NAME is the identifier for the textual region from which a specified segment or block of text will be extracted.

FILE_NAME is the identifier for the ASCII or color-attributed file to be inserted into the textual region.

ASCII_FILE_NAME is the identifier for the ASCII file to be inserted into the list box.

*Use of
Keywords*

... at cursor ...

Specifies that the text is to be inserted at the cursor position.

... after cursor ...

Specifies that the text is to be inserted one column to the right of the cursor position.

... segment ...

Copies a segment of contiguous characters from one textual region into another.

... block ...

Copies a block from one textual region into another.

... file FILE_NAME

Inserts an ASCII text file into a textual region.

... colored file FILE_NAME

Inserts a color-attributed file into a textual region.

... file ASCII_FILE_NAME

Inserts an ASCII text file into a list box. If the list box is sorted, the lines of the specified text file are inserted in ascending or descending order. Otherwise, the lines are added to the end of the list.

Remarks:

For textual regions, you can insert a string value, a specified segment or block of text from another textual region, or an ASCII or color-attributed file. If text is inserted where there is no text, EASEL OS/2 EE will insert any necessary blanks up to the text cursor before doing the insert.

Valid control characters are **\b** (or **^H**), **\t** (or **^I**), and **\n** (or **^J**). All other control characters are ignored. When **\t** is used, it will be expanded to spaces up to the next tab stop.

For textual regions, **insert at cursor ...** inserts the text string or segment beginning at the current position of the text cursor. All text after the inserted text is moved to the right (or down, if the text contains one or more new lines). After the insertion, the text cursor is positioned on the first character following the last inserted character. This is the same character on which the text cursor was positioned before the insertion, but it is now in a new position.

insert

For textual regions, **insert after cursor ...** inserts the text string or segment beginning one column to the right of the text cursor. All text after the inserted text is moved to the right (or down, if the text contains one or more new lines). After the insertion, the text cursor is positioned on the character one column to the left of the first inserted character. This is the same character on which the text cursor was positioned before the insertion, and the same position.

For textual regions, blocks are inserted at the current cursor position.

For sorted list boxes, **TEXT_STRING** inserts a string into its sorted position, and all text after the inserted string is moved down. For unsorted list boxes, **TEXT_STRING** inserts a string at the end of the list.

Examples: textual region Writer window size 30 columns 10 lines
at position 10 10
insert "Now is the time:" time

textual region Writer window size 30 columns 10 lines
at position 10 10
insert segment 3 1 thru 5 10 from OldText

textual region Writer window size 30 columns 10 lines
at position 10 10
insert block 5 10 thru 5 19 from OldText

textual region Writer window size 30 columns 10 lines
at position 10 10
insert file "sales.txt"

response to AddInfo
add to InfoTextregion
insert "More Information"

enabled visible multiple selection list box ListBox2
size 100 127 at position 30 42 in DialogBox2
insert "Choice 1\nChoice 2\nChoice 3"

enabled visible multiple selection list box ListBox2
size 100 127 at position 30 42 in DialogBox2
insert file "choices.fil"

See Also: **block** Definition (for Text)
read Action Statement
segment Definition

invisible Attribute Definitions

See: **visible** and **invisible** Attribute Definitions on page 1-356.

invoke Action Statement

Purpose: To execute another application.

Model

invoke PROGRAM_NAME [COMMAND_LINE_OPTIONS]

Where: PROGRAM_NAME is a string value representing the OS/2 filename of the application to be executed. You must specify the full filename, including the extension (which is always ".exe").

COMMAND_LINE_OPTIONS is an optional string value representing any options required by the program.

Remarks: EASEL OS/2 EE continues to run while the invoked program is running.

If starting a non-PM application, the application runs in its own fullscreen OS/2 session (text mode).

Examples: invoke "textedit.exe" "c:\\document\\memo5.doc"

ioerror Action Inquiry Built-in Function

Purpose: To test whether a **read** or **write** action statement, or an **insert file** drawing statement, was executed successfully.

Model

ioerror

Remarks: **ioerror** is a boolean function, returning the value **false** if the operation was successful, and **true** if it was not.

If the operation was not successful, **ioerror** is set to **true** and an error message is generated.

Default Values: **false**

Examples: if (ioerror) then
 :
 end if

See Also: **insert** Drawing Statement
read Action Statement
write Action Statement

is checked Object/Item Inquiry Built-in Function

Purpose: To find out the current check state of a radio button, check box, or choice item.

Model #1 (for Radio Buttons and Check Boxes):

OBJECT_NAME is checked

Model #2 (for Choices)

(item ITEM_NAME [from AB_NAME] in REGION_NAME is checked)

Where: OBJECT_NAME is the identifier for a radio button or check box.

ITEM_NAME is the name of an item.

AB_NAME is the name of an action bar template. You must specify the action bar name if more than one template contains the named item.

REGION_NAME is the name of a region containing the item.

Remarks: The value returned is the boolean (**true** or **false**) associated with the check state of the specified object or item. If the object or item is checked, the value **true** is returned; if it is unchecked, the value **false** is returned.

For items, the entire statement must be enclosed in parentheses.

Examples:

```
response to OK
  if (Copy_To_All is checked) then
    check Copy_To_Disk
    check Copy_To_Network
  end if

response to Make_Graph
  if (item Bar from View_Graph_Window_AB in
    View_Graph_Window is checked) then
    action GBars
  end if
```

See Also: **check** and **uncheck** Action Statements

is enabled Item Inquiry Built-in Function

Purpose: To inquire if an item is enabled (selectable).

Model

(item ITEM_NAME [from AB_NAME] in REGION_NAME is enabled)

Where: ITEM_NAME is a string value representing the identifier for an item.

AB_NAME is a string value representing the identifier for an action bar template. You must specify the action bar name if more than one template contains an item of the given name.

REGION_NAME is a string value representing the identifier for a region containing the item.

Remarks: The entire statement must be enclosed in parentheses.

The value returned is the boolean (**true** or **false**) associated with the enabled state of the specified object. If the item is enabled, the value **true** is returned; if it is disabled, the value **false** is returned.

Examples: response to item Exit from FileActionBar
if (item Open from FileActionBar in MainRegion is enabled) then
send "user has closed file\n" to errorlog
exit
end if

item Reference

Purpose: To reference an item defined outside of an action bar template.

Model

item ITEM_NAME

Where: ITEM_NAME is the identifier for a pulldown, button, choice, or separator previously defined outside of an action bar template.

Remarks: Only items defined outside of an action bar template can be referenced in an action bar template.

Examples:

```
# This is a pulldown choice defined outside of an
# action bar template
checked choice OpenFile text "~Open File"

# This is the Action Bar Template
action bar MainMenu is
    pulldown FileMenu text "File"
        item OpenFile      # Item ref to predefined choice
    :
    end pulldown
    :
end action bar
```

See Also: **button** Definition
choice Definition
pulldown Definition
separator Definition

item class Definition

Purpose: To group a number of items under one identifier. This allows you to specify one action statement or one response definition for a group of items.

Model

```
item class ITEM_CLASS_NAME [is
    ITEM_NAME1 [from AB_NAME1] [ITEM_NAME2 [from AB_NAME2] ]... ]
```

Where: ITEM_CLASS_NAME is the identifier for a class containing items.

ITEM_NAME1 and ITEM_NAME2 are identifiers for a pulldown, button, choice, or separator.

AB_NAME1 and AB_NAME2 are identifiers for an action bar template. The action bar name must be specified if more than one template contains an item of the given name. It cannot be specified if the item was defined outside of an action bar template.

Remarks: An item class can only contain items.

An item class is always referenced with the word **item** before the word **class**.

All the items in the item class definition must have been previously defined in the program. The class can be initially null.

As a general rule, it is inadvisable to group different types of items as members of the same class. You should define classes that consist of only one type of item.

You cannot define a class as a member of another class; only individual items can be defined as class members.

An item can be a member of more than one class.

Once a class is defined, you can reference it in any action statement that is valid for an item. When the actions are executed, they will affect all the items in the class. Also, you can define a response to the class, and the response will be invoked by any of the items in the class.

Examples: item class AllItemClass

```
item class FileClass is
  Open
  Save
  Exit
```

```
item class ItemClass is
  Open from Master_ActionBar
  Save from Master_ActionBar
  Exit from Master_ActionBar
```

See Also: **add to item class** Action Statement
 delete from item class Action Statement

key Definition

Purpose: To define a key.

Model

```
[ enabled | disabled ] [ visible | invisible ] [ [color] COLOR]  
key KEY_NAME  
  at { [position] X Y | default position }  
  [ in PARENT_NAME1 [ in PARENT_NAME2 ] ... ]  
  [ priority is PRIORITY_NO ]  
  [ parameter is PAR_STRING ]  
  
  [DRAWING_STATEMENT] ...
```

Where: COLOR is a string or integer value representing an EASEL OS/2 EE color.

KEY_NAME is the identifier for a key.

X and Y are integer values representing the position of the key.

PARENT_NAME1 and PARENT_NAME2 are identifiers for parent objects.

PRIORITY_NO is an integer value representing the priority of the object.

PAR_STRING is a string value representing the parameter of the object.

DRAWING_STATEMENT is a drawing statement.

Remarks: When specifying attributes, the **priority** and **parameter** can be specified in any order. Other attributes must be specified in the order shown in the model. Only the **key** specification, the identifier, and the **at** X Y specification are required. Other attributes are optional, using the defaults shown below.

A key cannot be the ancestor of any object. A key can contain any number of drawing statements.

A key cannot be a child of **desktop**.

Default Values: SELECTABILITY: **enabled**
 VISIBILITY: **visible**
 COLOR: **white**
 ANCESTRY: **in primary region**
 PARAMETER: **""**
 PRIORITY: **0**

Examples: key Start at position 100 200
 move 20 20
 box 30 30

disabled red key Times at default position in CalcPic
parameter is "Times"
box 22 22
move 8 6
text "*"

red key Shape at position 100 100
box 150 150

<i>See Also:</i>	color Keyword	position Keyword
	enabled and disabled	priority is Attribute
	Attribute Definitions	Definition
	parameter is Attribute	visible and invisible
	Definition	Attribute Definitions

Keyboard Mapping Table

Remarks: The table below describes the character codes generated within EASEL OS/2 EE for various keys and key combinations typed on the keyboard. EASEL OS/2 EE can respond to these character codes via the **response to char from keyboard** and **response to line from keyboard** response definitions.

Key or Key Combination	Character(s) Generated
Backspace	"\b" (or "\^H")
Enter	"\n" (or "\^J")
Esc	"\e\e" (or "\^[^[")
Ctrl + @	"\^C@"
Ctrl + A	"\^A"
Ctrl + B	"\^F\^B"
Ctrl + C	"\^F\^C"
Ctrl + D	"\^F\^D"
Ctrl + E	"\^F\^E"
Ctrl + F	"\^F\^F"
Ctrl + G	"\^G"
Ctrl + H	"\^H" (or "\b")
Ctrl + I	"\^I" (or "\t") (see <i>Exceptions</i>)
Ctrl + J	"\^J" (or "\n")
Ctrl + K	"\^K" (or "\v")
Ctrl + L	"\^L" (or "\f")
Ctrl + M	"\^M" (or "\r")
Ctrl + N	"\^N"
Ctrl + O	"\^O"
Ctrl + P	"\^P"
Ctrl + Q	"\^Q"
Ctrl + R	"\^R"
Ctrl + S	"\^S"
Ctrl + T	"\^T"
Ctrl + U	"\^U"
Ctrl + V	"\^V"
Ctrl + W	"\^W"
Ctrl + X	"\^X"
Ctrl + Y	"\^Y"
Ctrl + Z	"\^Z"
Ctrl + [	"\^[\[[" (or "\e\e")
Ctrl + \	"\^[\"
Ctrl +]	"\^]\"
Ctrl + ^	"\^^\"

Key or Key Combination	Character(s) Generated
Ctrl + _	" ^ "
Tab	" ^ " (or " \ t ") (see <i>Exceptions</i>)
BackTab	" \ ^ B T "
Alt + A	" \ ^ D A "
Alt + B	" \ ^ D B "
Alt + C	" \ ^ D C "
Alt + D	" \ ^ D D "
Alt + E	" \ ^ D E "
Alt + F	" \ ^ D F "
Alt + G	" \ ^ D G "
Alt + H	" \ ^ D H "
Alt + I	" \ ^ D I "
Alt + J	" \ ^ D J "
Alt + K	" \ ^ D K "
Alt + L	" \ ^ D L "
Alt + M	" \ ^ D M "
Alt + N	" \ ^ D N "
Alt + O	" \ ^ D O "
Alt + P	" \ ^ D P "
Alt + Q	" \ ^ D Q "
Alt + R	" \ ^ D R "
Alt + S	" \ ^ D S "
Alt + T	" \ ^ D T "
Alt + U	" \ ^ D U "
Alt + V	" \ ^ D V "
Alt + W	" \ ^ D W "
Alt + X	" \ ^ D X "
Alt + Y	" \ ^ D Y "
Alt + Z	" \ ^ D Z "
F1	" \ ^ F 1 " (see <i>Exceptions</i>)
F2	" \ ^ F 2 "
F3	" \ ^ F 3 "
F4	" \ ^ F 4 "
F5	" \ ^ F 5 "
F6	" \ ^ F 6 "
F7	" \ ^ F 7 "
F8	" \ ^ F 8 "
F9	" \ ^ F 9 "
F10	" \ ^ F 0 " (see <i>Exceptions</i>)
F11	" \ ^ F "
F12	" \ ^ F + "
Shift + F1	" \ ^ B 1 "
Shift + F2	" \ ^ B 2 "
Shift + F3	" \ ^ B 3 "
Shift + F4	" \ ^ B 4 "
Shift + F5	" \ ^ B 5 "
Shift + F6	" \ ^ B 6 "

Keyboard Mapping Table

Key or Key Combination	Character(s) Generated
Shift + F7	"\^B7"
Shift + F8	"\^B8"
Shift + F9	"\^B9"
Shift + F10	"\^B0"
Shift + F11	"\^B_"
Shift + F12	"\^B +"
Ctrl + F1	"\^C1"
Ctrl + F2	"\^C2"
Ctrl + F3	"\^C3"
Ctrl + F4	"\^C4"
Ctrl + F5	"\^C5"
Ctrl + F6	"\^C6"
Ctrl + F7	"\^C7"
Ctrl + F8	"\^C8"
Ctrl + F9	"\^C9"
Ctrl + F10	"\^C0"
Ctrl + F11	"\^C_"
Ctrl + F12	"\^C +"
Alt + F1	"\^D1"
Alt + F2	"\^D2"
Alt + F3	"\^D3"
Alt + F4	"\^D4" (see <i>Exceptions</i>)
Alt + F5	"\^D5" (see <i>Exceptions</i>)
Alt + F6	"\^D6" (see <i>Exceptions</i>)
Alt + F7	"\^D7" (see <i>Exceptions</i>)
Alt + F8	"\^D8" (see <i>Exceptions</i>)
Alt + F9	"\^D9" (see <i>Exceptions</i>)
Alt + F10	"\^D0" (see <i>Exceptions</i>)
Alt + F11	"\^D_"
Alt + F12	"\^D +"
Home	"\eH"
Up Arrow	"\eU"
Down Arrow	"\eD"
Left Arrow	"\eL"
Right Arrow	"\eR"
Page Up	"\^FT"
Page Down	"\^FB"
End	"\^FE"
Insert	"\^FI"
Delete	"\^FX"
Ctrl + Up Arrow	"\^CU"
Ctrl + Down Arrow	"\^CD"
Ctrl + Left Arrow	"\^CL"
Ctrl + Right Arrow	"\^CR"
Ctrl + Page Up	"\^CT"
Ctrl + Page Down	"\^CB"
Ctrl + Home	"\^CH"

Key or Key Combination	Character(s) Generated
Ctrl + End	"\^CE"
Ctrl + Print Screen	"\^C*"
Alt + 1	"\^D!"
Alt + 2	"\^D@"
Alt + 3	"\^D
Alt + 4	"\^D\$"
Alt + 5	"\^D% "
Alt + 6	"\^D^"
Alt + 7	"\^D&"
Alt + 8	"\^D*"
Alt + 9	"\^D("
Alt + 0	"\^D)"
Alt + -	"\^D-"
Alt + =	"\^D= "

Note: For character codes containing two characters, you need to respond to each character separately. (See *Examples.*)

Exceptions: For Ctrl-I and Tab, if you are compiling with the CONFIG.ESL keyword **KBDTABWID** = 0, the character generated is as shown in the table. Otherwise, the characters generated are the number of spaces required to bring the current column position to the next tab stop, where the width of the tab stop is set in **KBDTABWID**.

For windowed applications, the following keys are reserved by PM and are unavailable for use in responses: F1, F10, Alt-F4 through Alt-F10.

Examples:

```
# Respond to ALT-letter combinations
# First, ALT-A is typed
# Then, ALT-B is typed

response to char "\^D" from keyboard      # ALT
begin guarded
    response to char "A" from keyboard    # ALT-A typed
    :
    response to char "B" from keyboard    # ALT-B typed
    :
end
```

See Also: Escape Sequences
CONFIG.ESL File (Chapter 3)

Keywords

Remarks: Keywords are reserved words in the EASEL OS/2 EE language.

You must specify all EASEL OS/2 EE keywords in lower-case, exactly as shown in the syntax models.

Examples: **key**
enabled

leave block Action Statement

Purpose: To explicitly leave (exit) a block.

Model

leave block [BLOCK_NAME]

Where: BLOCK_NAME is the identifier for the block.

Remarks: The **leave block** statement can be specified inside a response definition that is inside a block. Note, however, that the statement must be specified within the definition of the block itself; it cannot be specified in an action routine that is called from inside the block.

EASEL OS/2 EE will exit the block when the **leave block** statement is executed. Execution continues at the action statement (if any) immediately following the **end** statement for the block in which the **leave block** statement is specified.

Once a block has been exited, the responses in that block become inactive and cannot be executed again unless the block is re-entered.

You have the option to specify a name in the **leave block** statement. This name specifies the outermost block that you wish to leave. EASEL OS/2 EE exits the block specified in the **leave block** statement, and all blocks within that block. If you do not specify a name in the **leave block** statement, EASEL OS/2 EE exits only the innermost active block.

Examples:

```

response to char from Remote
  begin
    response to ...
  :
    response to ...
    leave block
  end
response to line "##" from Remote
  begin ChoiceA
    response to ...
    leave block ChoiceA
  end ChoiceA

```

leave blank

See Also: **begin** BLOCK Action Statement

leave loop Action Statement

Purpose: To explicitly leave (exit) a loop.

Model

leave loop [LOOP_NAME]

Where: LOOP_NAME is the identifier for the loop.

Remarks: The **leave loop** action statement can be one of the action statements in a **for each member loop**, **for each selected line loop**, **for loop**, or **while loop** statement.

Upon execution, EASEL OS/2 EE will exit from the loop before the condition specified in the **for** or **while** statement has failed, and will continue execution at the statement following the **end loop** keywords.

If you specify a **leave loop** statement, you must specify it within the definition of the loop itself. You cannot specify it in an action routine that is called from inside the loop.

You can specify a name in the **leave loop** statement. This name might identify the loop in which the **leave loop** is specified, or it might identify an outer loop. If you specify an outer loop, EASEL OS/2 EE leaves the specified outer loop and all inner loops. If you do not specify a name in the **leave loop** statement, EASEL OS/2 EE exits from the innermost loop that contains the **leave loop** statement, even if the innermost loop has a name.

Examples:

```
response to Help2
  while (Int <= 5) loop
    action Action_Sub
    if (Action_flag != "OK") then
      leave loop
    end if
    copy (Int + 1) to Int
  end loop
  copy Action_flag to Flag_log
```

See Also: **for each member loop** Action Statement
for each selected line loop Action Statement
for loop Action Statement
while loop Action Statement

left [of] and right [of] Object Inquiry Built-in Functions

Purpose: To find and use the current coordinate positions describing the smallest rectangular area occupied by an object, its contents, children, and children's contents.

Model

{ left | right } [of] OBJECT_NAME

Where: OBJECT_NAME is a string value representing the identifier for an object.

Remarks: The values returned are based upon the specified object's contents (drawing statements), children, and children's contents, and are expressed in the coordinates of the object. If the object has no contents, EASEL OS/2 EE returns values of zero for these functions.

For a graphical object and dialog region, the following values are returned:

left [of] The leftmost X coordinate occupied by the contents of the object.

right [of] The rightmost X coordinate occupied by the contents of the object.

For an image region, the following values are returned:

left [of] Always 0.

right [of] The rightmost X coordinate occupied by the image file currently displayed in the region.

For a textual region, the following values are returned:

left [of] Always 1. This is the column number of the first column in the textual region.

right [of] The column number of the last column of the longest line in the textual region.

For an entry field, the following values are returned:

left [of] Always **1**. This is the column number of the first character in the entry field.

right [of] The column number of the last character in the entry field.

Examples: response to Move
if ((right of Zoom) < MaxZ)
then add to Zoom
move to ((right of Zoom) + 10) Height

See Also: **top [of]** and **bottom [of]** Object Inquiry Built-in Functions
xmiddle [of] and **ymiddle [of]** Object Inquiry Built-in Functions

length of Special Inquiry Built-in Function

Purpose: To determine the number of characters in a specified string variable.

Model

length of VAR_NAME

Where: VAR_NAME is the identifier for a string variable.

Remarks: The value returned is an integer, representing the number of characters, including spaces, in the specified string variable.

Examples: response to keyboard
if ((length of input) > 20) then
change ErrorReg to
"Input Too Long"
end if

line size of Object Inquiry Built-in Function

Purpose: To determine the height of a line in a specified textual region.

Model

line size of TR_NAME

Where: TR_NAME is the identifier for the textual region.

Remarks: This function can be used only with textual regions.

line size of returns an integer representing the height of a character, plus the character's associated vertical intercell gap, in the coordinates of the textual region's parent.

Examples: response to Mover
change Logo in CoData position to
(xcursor of Start) * line size of Start) 100

See Also: **column size of** Object Inquiry Built-in Function

list box Definition

Purpose: To define a list box.

Model

```
[ enabled | disabled ] [ visible | invisible ]  
[multiple selection] list box LB_NAME  
    size WIDTH HEIGHT  
    at [position] X Y  
    in DB_NAME  
    [parameter is PAR_STRING]  
    [sort { ascending | descending } ]  
    [horizontal scroll bar]  
    [group is GROUP_NAME]  
    [insert { TEXT_STRING | file ASCII_FILE_NAME } ]...
```

Where: LB_NAME is the identifier for a list box.

WIDTH and HEIGHT are integer values representing the dimensions of the list box in dialog units.

X and Y are integer values representing the position of the list box in dialog units.

DB_NAME is the identifier for the dialog box or dialog region parent object.

PAR_STRING is a string value representing the parameter of the list box.

GROUP_NAME is an identifier for the group in which the list box belongs.

TEXT_STRING is a string value representing the text to be inserted into the list box.

ASCII_FILE_NAME is the identifier for the ASCII file to be inserted into the list box.

Use of **multiple selection**
Keywords: Specifies that this list box allows the user to make multiple selections.

sort ...

Specifies that the list box entries are to be sorted in ascending or descending order. If sort is not specified, then entries inserted into the list box will be added to the end.

horizontal scroll bar

Specifies that the list box will contain a horizontal scroll bar.

group is ...

Specifies that this list box is to be grouped with all other dialog control objects which specify this group name.

Remarks: A list box is a dialog control, and it must have a dialog box or dialog region as its parent.

Only the **list box** keywords, the identifier, the **at** [**position**] specification, the **in** DB_NAME, and the **size** specification are required in the definition.

When specifying attributes, the **parameter**, **sort**, **horizontal scroll bar**, and **group is** can be specified in any order; all other attributes must be specified in the order shown in the model.

<i>Default</i>	SELECTABILITY:	enabled
<i>Values:</i>	VISIBILITY:	visible
	MULTIPLE SELECTION:	no
	PARAMETER:	""

Examples: invisible dialog box SaveFile
size 160 100 at position 200 200
dialog border
title bar "Save Output File"
system menu

```
list box Directories
  size 60 50 at position 6 20 in SaveFile
  sort ascending
  insert "[sources]"
  insert "[documents]"
```

See Also: **combination box** Definition
dropdown list Definition
for each selected line loop Action Statement
remove Action Statement
selected line from Object Inquiry Built-in Function

make COLOR Action Statement

Purpose: To change the foreground color of keys, graphical regions, image regions, or textual regions; to change the background color of graphical regions, image regions, or textual regions.

Model

```
make { OBJECT_NAME | CLASS_NAME }  
[ background | foreground ] [color] COLOR
```

Where: OBJECT_NAME is the identifier for a key, graphical region, image region, or textual region.

CLASS_NAME is the identifier for a class of keys, graphical regions, image regions, or textual regions.

COLOR is a string or integer value representing an EASEL OS/2 EE color.

Remarks: By default, this statement changes the foreground of a key and the background of a region.

Changing the foreground of a graphical object changes the default color of the object's drawing statements, but it does not override the explicit colors specified within the drawing statements.

Examples: response to Ocean
 make object aqua

response to Crustacean
 make Lobster in Seafood color ColorChoice

See Also: **color** Keyword
 make border COLOR Action Statement

make block and make segment COLOR Action Statements

Purpose: To specify colors for individual characters, or a group of characters, in a colored textual region.

Model

```
make CTR_NAME { block | segment } [column] COL_NO [line] LINE_NO
               [thru [column] COL_NO [line] LINE_NO]
               [foreground [color] COLOR]
               [background] [color] COLOR
```

Where: CTR_NAME is the identifier for a colored textual region.
 COL_NO is an integer value representing the column numbers of the first and last characters to be colored.
 LINE_NO is an integer value representing the line numbers of the first and last characters to be colored.
 COLOR is an integer value or a character string representing the foreground or background color.

Use of Keywords: **make segment ...**
 Assigns a color to a text segment (see **segment Definition**). To color two or more text segments, write a series of **make segment** statements.

make block ...
 Assigns a color to a text block (see **block Definition (for Text)**). To color two or more text blocks, write a series of **make block** statements.

Remarks: The **make segment** and **make block** action statements allow you to assign colors to individual characters, or groups of characters, in a colored textual region, only when the colored textual region in the EASEL OS/2 EE program already contains the text to be colored.

make block and make segment

Examples:

```
response to Add2
  make CTRegion_A
    segment column 1 line 1
    foreground red
    background black

response to Hilite
  make Weather
    segment column 1 line 1 red

response to Hilite
  make News
    segment column 1 line 1 thru column 2 line 2
    foreground red blue

response to Hilite
  make Weather
    block column 5 line 60 thru column 5 line 80
    foreground color 18 color 24
```

See Also: **block** Definition (for Text)
 segment Definition

make border COLOR Action Statement

Purpose: To change the color of the border in a graphical, image, or textual region.

Model

```
make { REG_NAME | REG_CLASS_NAME } border [color] COLOR
```

Where: REG_NAME is the identifier for a graphical, image, or textual region.

REG_CLASS_NAME is the identifier for a class of graphical, image, or textual regions.

COLOR is a string or integer value representing a color.

Remarks: If the **border** attribute is not specified in the object definition, this statement will have no effect unless the border is first made visible.

This statement does not change the color of PM borders.

This statement has no effect on regions with frame attributes.

Examples: response to GraphArea
 make GraphArea border red

response to Select
 make LineGraph in GraphArea
 border color HighlightColor

See Also: **border** Attribute Definition
make border invisible and **make border visible** Action Statements

make border invisible and make border visible Action Statements

Purpose: To change the visibility of the border in a graphical, image, or textual region.

Model

```
make { REG_NAME | REG_CLASS_NAME } border { visible | invisible }
```

Where: REG_NAME is the identifier for a graphical region, image region, or textual region.

REG_CLASS_NAME is the identifier for a class of graphical regions, image regions, or textual regions.

Remarks: If the **border** attribute is not specified in the object definition, this statement will add the border to the object and make it visible (and, by default, the color white).

This statement has no effect on regions with frame attributes.

Examples: response to SelectYes
 make Area2 border invisible

response to Menu
 make Area1 in Area2 border visible

See Also: **make** COLOR Action Statement

make border visible Action Statement

See: **make border invisible** and **make border visible** Action Statements on page 1-216.

make cursor Action Statement

Purpose: To control the visibility of the text cursor in a textual region.

Model

make { TR_NAME | TR_CLASS_NAME } cursor { visible | invisible }

Where: TR_NAME is the identifier for a textual region.
TR_CLASS_NAME is the identifier for a class of textual regions.

Remarks: This statement can be used for textual regions only.
The text cursor of a textual region is, by default, invisible. When the text cursor is made visible, it appears as a rectangle surrounding the character position on which it lies.
The text cursor is drawn in the foreground color of the textual region. If the textual region is colored, the text cursor is drawn in the foreground color of the character on which it lies.

Examples: response to ShowMe
 make WriteHere cursor visible

make invisible and make visible Action Statements

Purpose: To change the visibility of an object, or of all the objects in a class.

Model

```
make { OBJECT_NAME | CLASS_NAME } { visible | invisible }
```

Where: OBJECT_NAME is the identifier for an object.
CLASS_NAME is the identifier for a class of objects.

Remarks: When a region with frame attributes is made visible, it is also activated.

Care should be taken when changing the visibility of dialog control objects. Note that unpredictable results may occur.

Examples: response to Time
 make Hourglass visible

 response to SetClock
 make Stopwatch in Timepieces invisible

See Also: **activate** Action Statement
make border invisible and **make border visible** Action Statements
visible and **invisible** Attribute Definitions
visibility of Object Inquiry Built-in Function

make parameter Action Statement

Purpose: To change the parameter of an object or item.

Model #1 (for Objects):

```
make { OBJECT_NAME | CLASS_NAME } parameter PAR_STRING
```

Model #2 (for Items):

```
make item { ITEM_NAME [from AB_NAME] | ITEM_CLASS_NAME }  
  parameter PAR_STRING
```

Where: OBJECT_NAME is the identifier for an object.
CLASS_NAME is the identifier for a class of objects.
PAR_STRING is a string value representing the new parameter to be associated with the object.
ITEM_NAME is the identifier for an item.
AB_NAME is the identifier for an action bar template.
ITEM_CLASS_NAME is the identifier for a class of items.

Remarks: The parameter of an item is attached to, and is an attribute of, the action bar template or the standalone item template. If the parameter is changed within an action statement, it is changed in the template definition and everywhere it is referenced.

Examples: response to Home
 make BlueKey parameter "Colored_Key"

 response to Headquarters
 make RedItem in BlueItem parameter "Chicago"

 response to Test
 make item File_Open from Master_ActionBar
 parameter "tempfile.ea1"

See Also: **parameter** Response Inquiry Built-in Function
parameter is Attribute Definition
parameter of Object Inquiry Built-in Function

make point disposition Action Statement

Purpose: To control EASEL OS/2 EE's processing of responses to input from a pointing device.

Model

```
make point disposition [ mouse | touchscreen ] { normal | drop }
```

Use of Keywords:

- ... **mouse**
Affects the disposition of the mouse input.
- ... **touchscreen**
Affects the disposition of the touchscreen input.
- ... **normal**
When EASEL OS/2 EE receives input from the pointing device, it responds.
- ... **drop**
When EASEL OS/2 EE receives input, it ignores the input. No input is taken until the disposition is returned to **normal**.

Remarks: This statement affects input from any specified pointing device.

Default Values: If **mouse** or **touchscreen** is not specified, the specified disposition affects *all* pointing devices.

Examples:

```
response to char "$$$" from Host
    make point disposition mouse drop

action UseAgain is
    :
    make point disposition normal
```

See Also: **make string disposition** Action Statement
open and **close** Action Statement

make segment

make segment COLOR Action Statement

See: **make block** and **make segment** COLOR Action Statements
on page 1-213.

make string disposition Action Statement

Purpose: To control EASEL OS/2 EE's processing of responses to input from an application or from the keyboard.

Model

```
make string disposition { keyboard | APPLICATION_NAME }
    { normal | drop | hold }
```

Where: APPLICATION_NAME is an identifier for an application program.

Use of Keywords:

- ... **normal**
Specifies that input received is to invoke the appropriate response. This is the default disposition.
- ... **drop**
Specifies that input received is to be discarded.
- ... **hold**
Specifies that input received is to be saved until a **make string disposition ... normal** statement is executed, at which time the saved inputs invoke the appropriate responses.

Examples: response to line from keyboard
 make string disposition Recorder hold

```
action Ignore is
:
    make string disposition Sales drop
```

See Also: **make point disposition** Action Statement

make visible

make visible Action Statement

See: **make invisible** and **make visible** Action Statements on page 1-219.

marker Special Built-in Function

See: **extract from** Action Statement on page 1-134.

member Special Inquiry Built-in Function

Purpose: To access the name of the current class element in a **for each member** loop.

Model

member

Remarks: EASEL OS/2 EE returns a string value representing the name (identifier) of the current member.

Examples:

```
copy false to InClass
for each member of CLASS loop
  if (member = OBJ1) then      #check to see if OBJ1 is in
    copy true to InClass      #this class
    leave loop
  end if
end loop
```

See Also: **for each member loop** Action Statement

members of Special Inquiry Built-in Function

Purpose: To find the identifiers for the current objects in a class.

Model

members of CLASS_NAME

Where: CLASS_NAME is the identifier for a class.

Remarks: The value returned by the **members** function is a string of names of all the objects in the specified class, separated by spaces. If any name in the class is ambiguous, EASEL OS/2 EE will return the object's entire ancestry for clarification.

If you specify ancestry when you add an object to a class, the **members** function will return the object name along with that ancestry.

Examples:

```

response to Change
  copy 100 to XPos
  extract from members of PrintColors
    take word PrintStr # Reset marker; get first
    object.
  change PrintStr position to XPos 40 in Print2
  copy (XPos + 100) to XPos
  while (marker < length of members of PrintColors)
  loop
    extract from members of PrintColors
    skip to marker
    take word PrintStr
  change PrintStr position to XPos 40 in Print2
  copy (XPos + 100) to XPos
end loop

```

Errors: If the class does not contain any objects, the **members** function returns the null string.

See Also: **class** Definition

module

module Reference

Purpose: To reference a library module.

Model

```
module { BGraph | CText | M3270 | M5250 }
```

Remarks: The **module** statement tells the EASEL OS/2 EE processor to make available, in the EASEL OS/2 EE program, the action routines that comprise the module. When you reference a module, each action statement that is part of the module is available in the program exactly as though you had specified its definition in the program itself. You can specify any number of module statements in an EASEL OS/2 EE program.

Examples: module M3270
 module BGraph

move Drawing Statement

Purpose: To move the text and graphics cursors to a specified position in a graphical object, or to move the text cursor to a specified position in a textual region.

Model #1 (for Graphical Objects):

```
move { [by] XOFFSET YOFFSET | to X Y }
```

Model #2 (for Textual Regions):

```
move { [by] COLUMNS [columns] LINES [lines] |  
      to [column] COL_NO [line] LINE_NO }
```

Where: XOFFSET is an integer value representing the number of coordinate positions to move in the X direction.

YOFFSET is an integer value representing the number of coordinate positions to move in the Y direction.

X is an integer value representing an absolute X coordinate position.

Y is an integer value representing an absolute Y coordinate position.

COLUMNS and LINES are positive or negative integer values representing the number of columns and lines (respectively) by which to move.

COL_NO and LINE_NO are positive integer values representing the column and line number to which to move.

*Use of
Keywords:*

move by ...

Specifies a relative position by which the cursor is to move. If negative values are used for textual regions, the resulting position must still be in the positive quadrant.

move to ...

Specifies an absolute position to which the cursor is to move. For textual regions, the position must be non-negative.

move

Remarks: For graphical objects, the **move** statement moves the graphics and text cursor to the specified position.

For textual regions, the **move** statement moves the text cursor to the specified position. Any position may be specified, whether text exists there or not.

Default Values: If neither **by** nor **to** is specified, EASEL OS/2 EE uses the specification **by**.

Examples:

```
textual region Writer window size 60 columns 35 lines
    at position 0 50
    move 23 columns 40 lines
    :

textual region Writer2 window size 10 columns 30 lines
    at position 0 50
    move to column 6 line 10
    :

graphical region Holder size 400 50 at position 10 10
    move -5 25
    :

graphical region Holder2 size 400 50 at position 10 10
    move to 5 25
    :
```

Errors: An error message will be generated if the result of a **move** [**by**] or **move to** statement for a textual region results in a zero or negative column or line number.

See Also: **draw** Drawing Statement

move arc Drawing Statement

Purpose: To move the graphics cursor in an arc, as part of a boundary specification within a shape definition.

Model #1 (in Specified Degrees):

```
move arc DEGREES degrees [ counter clockwise | clockwise ]
      center [ by | to | at ] XCENTER YCENTER
```

Model #2 (to a Specified Endpoint):

```
move arc { by | to } XEND YEND [ counter clockwise | clockwise ]
      center [ by | to | at ] XCENTER YCENTER
```

Where: DEGREES is an integer value representing the number of degrees of the arc.

XCENTER is an integer value representing the horizontal center of the arc.

YCENTER is an integer value representing the vertical center of the arc.

XEND is an integer value representing the horizontal endpoint of the arc.

YEND is an integer value representing the vertical endpoint of the arc.

Use of Keywords:

- ... degrees DEGREES**
Moves the graphics cursor in an arc by the specified number of degrees. Positive degrees moves the arc in the direction of travel around the circle. Negative degrees moves the arc opposite to the direction of travel around the circle.
- ... counter clockwise**
Defines the direction of travel around the circle to be counterclockwise. This is the default.
- ... clockwise**
Defines the direction of travel around the circle to be clockwise.

move arc

... **center** [**by**]

Specifies the center of the arc as a position relative to the current position of the graphics cursor.

... **center to** or **center at**

Specifies the center of the arc as an absolute position in the coordinate system of the object. The keywords **to** and **at** are synonymous.

... **by** XEND YEND

Moves the graphics cursor in an arc by the specified number of coordinate positions relative to the current position of the graphics cursor. If the starting and ending points of the arc are not the same distance from the center and a **center** is specified, the cursor is moved from the starting point and the center exactly as specified, with the endpoint of the arc as close as possible to the specified endpoint while preserving the circularity of the arc.

... **to** XEND YEND

Moves the graphics cursor in an arc to the absolute coordinate position specified by XEND and YEND. If the starting and ending points of the arc are not the same distance from the center and a **center** is specified, the cursor is moved in an arc from the starting point and the center exactly as specified, with the endpoint of the arc as close as possible to the specified endpoint while preserving the circularity of the arc.

Remarks: The **move arc** statement can only be used within a **boundary** specification within a **shape** drawing statement. This statement moves the graphics cursor from its current position to a new position along an arc, without drawing anything.

The **center** specification defines the center of the circular boundary defined by the arc. You must provide a **center** specification for the first arc in the first **shape** statement in a graphical object. Thereafter, if you specify a series of **move arc** and **draw arc** statements in a single graphical object, and if each arc has the same center, you do not need to provide additional **center** specifications. If you do not specify any **center** specification in one of these **move arc** or **draw arc** statements, the center is assumed to be the same as the center of the previous arc in that graphical object.

Default Values: If you omit both the **clockwise** and the **counter clockwise** keywords, the direction of travel around the circle is counterclockwise.

Examples: disabled key Logo at position 10 10
:
move arc 50 degrees center at 100 0
:

key Help at position XPos YPos
:
move arc 90 degrees clockwise center by -50 150
:

graphical region Shaper size 100 100 at position 0 0
:
move arc by 45 90 center by XC YC
:

See Also: **draw arc** Drawing Statement
shape Drawing Statement

multiline entry field Definition

Purpose: To define a multiple-line entry field.

Model

```
[ enabled | disabled ] [ visible | invisible ] [ [color] COLOR]
multiline entry field EF_NAME
  size WIDTH HEIGHT
  at [position] X Y
  in DB_NAME
  [ [color] COLOR foreground]
  [parameter is PAR_STRING]
  [ignore tab]
  [word wrap]
  [ text { width XLIM | height YLIM } |
    text width XLIM height YLIM ]
  [text size TEXT_LIMIT]
  [horizontal scroll bar]
  [vertical scroll bar]
  [group is GROUP_NAME]
  [font FONT_NAME]
  [text TEXT_STRING]
```

Where: COLOR is a string or integer value representing a color.

EF_NAME is the identifier for the entry field.

WIDTH and HEIGHT are integer values representing the dimensions of the entry field in dialog units.

X and Y are integer values representing the position of the entry field in dialog units.

DB_NAME is the identifier for the dialog box or dialog region parent object.

PAR_STRING is a string value representing the parameter of the entry field.

XLIM and YLIM are integer values representing the maximum horizontal and vertical text extent in pixels.

TEXT_LIMIT is the maximum text length in bytes.

GROUP_NAME is a string value identifying the group to which the entry field belongs.

FONT_NAME is a string value representing the font to be used in the entry field.

TEXT_STRING is a string value representing the initial value that the entry field will contain.

*Use of
Keywords:*

ignore tab

Specifies that the entry field will ignore the TAB key. (This will allow the use of the TAB key to move to the next control in the dialog box or dialog region, and is required by CUA.)

word wrap

Specifies that the system will automatically insert a new line when maximum text width is reached. The maximum text width is the end of the visible field if no **text width** is specified. Otherwise, the maximum text width is the value specified in the **text width** attribute. Note that spaces are allowed to extend past the maximum text width.

text width ... height ...

Specifies the maximum extent of the text entered into the entry field, expressed in pixels. By default, the maximum text extent is defined by the window size if the multiple-line entry field has no scroll bar. Otherwise, the text extent is unlimited.

text size ...

Specifies the maximum text length in bytes.

horizontal scroll bar

Specifies that the entry field will contain a horizontal scroll bar.

vertical scroll bar

Specifies that the entry field will contain a vertical scroll bar.

group is ...

Specifies that this entry field is to be grouped with all other dialog control objects using the specified group name.

font ...

Specifies the font to be used for displaying the entry field contents. If not specified, **system** font is used.

text ...

Specifies the initial text that the entry field will contain.

multiline entry field

Remarks: A multiple-line entry field is a dialog control and must have a dialog box or dialog region as its parent.

Only the **multiline entry field** keywords, the identifier, the **at [position]** specification, the **in DB_NAME**, and the **size** specification are required, and they must be specified in the order shown in the model. Other attributes are optional, using the defaults shown below.

Any of the attributes after ancestry can be specified in any order.

Default	SELECTABILITY:	enabled
Values:	VISIBILITY:	visible
	COLOR (background)	black
	COLOR (foreground)	white
	PARAMETER:	""
	TEXT:	""
	FONT:	system

Examples: multiline entry field MessageText
size 150 80 at 25 10
vertical scroll bar
word wrap

See Also: **entry field** Definition

object Response Inquiry Built-in Function

Purpose: To find the name of the object or item that stimulated the current response.

Model

object

Remarks: The **object** function should be specified only within a **response to OBJECT** or **response to item** response definition.

EASEL OS/2 EE returns a string value representing the name (identifier) of the object or item that was selected. The name is obtained from the object or item definition. The ancestry is not returned.

If the **object** function is specified more than once within a single response, the same value will be returned until the response is completed.

Examples:

```
response to HelpClass
  make object red
  if (object = "StartKey") then
  :
  end if
```

Errors: If the function is specified within any response other than a **response to OBJECT** or **response to item**, EASEL OS/2 EE will return the null string as the value of the function.

See Also: **response to OBJECT** Response Definition
response to item Response Definition

Object Types and Ancestry

Remarks: The following are all the EASEL OS/2 EE object types and their valid parent types. The object type and associated valid parent types listed are the only ancestry combinations that can be specified.

Object Type	Valid Parent Types
dialog box	desktop
Dialog Controls	dialog region and dialog box
dialog region	desktop , graphical region , and image region
graphical region	desktop , graphical region , and image region
image region	desktop , graphical region , and image region
key	graphical region and image region
sense region	graphical region and image region
textual region	desktop , graphical region , and image region

Dialog controls are specified in "Dialog Control Objects" on page 1-101.

Graphical objects are of type: **graphical region** and **key**.

open and close Action Statement

Purpose: To enable or disable the use of a mouse or touchscreen as an alternate pointing device.

Model

```
{ open | close } { mouse | touchscreen }
```

Remarks: If a device is the default device, it is already open.

Usually, **open** is part of a **response to start** response definition.

If an alternate device is still open when EASEL OS/2 EE encounters an **exit** statement, the device is implicitly closed. If the device is open when EASEL OS/2 EE encounters a **change to program** statement, the device will remain open for the new EASEL OS/2 EE program.

The touchscreen does not display a pointer.

The **close** statement closes an alternate pointing device that has been opened. Once a device has been closed, the device can no longer be used for selecting enabled graphical objects or items.

Default Values: All pointing devices, except for the default device, are closed unless explicitly opened with the **open** statement.

Examples:

```
response to UseTouch
    close mouse
    open touchscreen

response to UseMouse
    open mouse
```

See Also: **response to start** Response Definition

overwrite Drawing Statement

Purpose: To overwrite text with a string value, or a text segment or text block from any textual region.

Model

```
overwrite { [at cursor] [text] STRING |  
            segment [column] COL_NO [line] LINE_NO  
                [thru [column] COL_NO [line] LINE_NO] from TR_NAME |  
            block [column] COL_NO [line] LINE_NO  
                [thru [column] COL_NO [line] LINE_NO] from TR_NAME }
```

Where: STRING is a string value representing the overwriting text.

COL_NO is an integer value representing the beginning and ending column numbers in the block or segment.

LINE_NO is an integer value representing the beginning and ending line numbers in the block or segment.

TR_NAME is the identifier for a textual region.

Use of **overwrite** STRING
Keywords:

Each character is replaced in succession. After execution, the text cursor is on the first character immediately following the last overwritten character.

If the text cursor is at the column after the last character in a line, the **overwrite** is performed as if it were an **insert** statement, and the line is extended.

... **segment** ...

Overwrites with the specified segment.

... **block** ...

Overwrites with the specified block. If the block contains character positions which do not contain any text, these positions contain blanks when the block overwrites text. After execution, the text cursor has not moved.

Remarks: These statements can be used only with textual regions.

The **overwrite** statement replaces text with the specified text string, segment, or block. The replacement begins at the current position of the text cursor.

If the text cursor is at a location where there is no text when the **overwrite** statement is encountered, EASEL OS/2 EE will insert the necessary blanks prior to the text cursor.

If you overwrite text in a colored textual region with a string, or a segment or a block from an uncolored textual region, the new text will take on the color specifications of the overwritten characters.

Examples:

```
response to Chalk
  change Writer to
  overwrite "cat"
```



```
response to Chalk
  change Writer to
  overwrite segment 3 1 thru 10 4 from OldText
```



```
response to Chalk
  change Writer to
  overwrite block 1 1 thru 7 4 from OldText
```

Errors: If the specified segment or block overlaps the text to be overwritten, nothing will be overwritten and an error message will be produced.

See Also: **block** Definition (for Text)
segment Definition

parameter Response Inquiry Built-in Function

Purpose: To find the string associated with the **parameter** attribute of an object or item that stimulated the current response.

Model

parameter

Remarks: The **parameter** function can be specified only within a **response to OBJECT** or **response to item** response definition.

Default Values: If there is no **parameter** attribute associated with the object or item, EASEL OS/2 EE will return the null string as the value of the function.

Examples:

```
response to Keys
  append parameter to AccumulatedInput
  ...
  if ((parameter = ".") or (parameter = " ")) then
    action Special
  else
    action Digit
  end if
```

Errors: If the function is specified within any response other than a **response to OBJECT** or **response to item**, EASEL OS/2 EE will return the null string as the value of the function.

See Also: **make parameter** Action Statement
parameter of Object Inquiry Built-in Function
parameter is Attribute Definition
response to OBJECT Response Definition
response to item Response Definition

parameter is Attribute Definition

Purpose: To specify a value for an object or item, that can later be examined and returned by the **parameter** or **parameter of** built-in functions.

Model

parameter is PAR_STRING

Where: PAR_STRING is a string value representing the parameter of the object or item.

Remarks: The parameter specification must appear within the object or item definition.

The **parameter** attribute does not affect the appearance of the object or item. It allows you to perform action statements based upon the specified parameter value.

Default Values: If no parameter is specified, the default is the null string ("").

Examples: key Northwest at position 10 10
parameter is "Boston"

key Midwest at position 10 40
parameter is "Chicago"

- See Also:*
- | | |
|------------------------------------|--|
| button Definition | image region Definition |
| check box Definition | key Definition |
| choice Definition | list box Definition |
| combination box Definition | multiline entry field Definition |
| dialog box Definition | parameter Response Inquiry Built-in Function |
| dialog region Definition | parameter of Object Inquiry Built-in Function |
| dropdown list Definition | pulldown Definition |
| entry field Definition | push button Definition |
| graphical region Definition | radio button Definition |
| group box Definition | sense region Definition |
| | static text Definition |
| | textual region Definition |

parameter of Object Inquiry Built-in Function

Purpose: To find and use the current parameter string associated with the parameter attribute of a specified object.

Model

parameter of OBJECT_NAME

Where: OBJECT_NAME is a string value representing the identifier for an object.

Remarks: The value returned for the **parameter of** function is the string value representing the parameter string attribute associated with the specified object.

This function is similar to the **parameter** response inquiry built-in function, which returns the parameter of the object that stimulated the response.

Default Values: If you specify the **parameter of** function for an object for which no parameter has been defined, the value returned is the null string.

Examples: response to line "to Delete" from Host
make (parameter of Delete) visible

See Also: **make parameter** Action Statement
parameter Response Inquiry Built-in Function
parameter is Attribute Definition

pattern Pattern Reference

Purpose: To reference a previously defined pattern in a drawing statement.

Model

```
[ [color] COLOR ] [scale [ by | to ] NUMERATOR:DENOMINATOR]
[rotate [ by | to ] DEGREES] pattern PATTERN_NAME
```

Where: COLOR is an integer or string value representing an EASEL OS/2 EE color.

NUMERATOR and DENOMINATOR are integer values representing a scale ratio.

DEGREES is an integer value representing the number of degrees by which the pattern is to be rotated.

PATTERN_NAME is the identifier for a pattern.

*Use of
Keywords:*

[color] COLOR

Draws, in the specified color, all the drawing statements in the pattern that do not already have a specified color. This attribute overrides any color specifications that otherwise would affect these drawing statements (such as the foreground color in the object definition or in a **make** COLOR action statement).

scale NUMERATOR:DENOMINATOR

Enlarges or shrinks the pattern by the ratio specified by the numerator and denominator. EASEL OS/2 EE determines the scale factor by dividing the numerator by the denominator. If the scale factor is less than 1, the pattern is shrunk. If the scale factor is greater than 1, the pattern is enlarged.

Both the original pattern and the scaled pattern originate at the pattern's reference point. A pattern is always drawn from that point, regardless of the scale factor.

If the pattern contains text, EASEL OS/2 EE moves the graphics cursor and the text cursor as specified by the **scale** specification, and the cursor positions dictate the starting position of the text string. The text itself is not scaled, except in the case of outline fonts.

The scalings specified for a pattern are cumulative when patterns are nested. The specified scaling is relative to any scaling currently in effect for the pattern.

rotate DEGREES

Rotates the pattern by the specified integer value representing the number of degrees. Both the original pattern and the rotated pattern originate from the pattern's reference point. A pattern is always drawn beginning at that point, regardless of the specified rotation. If the DEGREES value is positive, the pattern is rotated counterclockwise. If it is negative, the pattern is rotated clockwise.

If the pattern is an ellipse, EASEL OS/2 EE rotates the ellipse to the nearest 90 degrees.

Avoid rotating a pattern that contains a **box** statement unless the pattern is rotated in 90-degree increments. Otherwise, rotating a box may cause unpredictable results.

If the pattern contains text, EASEL OS/2 EE moves the graphics cursor and the text cursor as specified by the **rotate** specification, and the cursor positions dictate the starting position of the text string. The text itself is not rotated; it is always displayed horizontally, from left to right.

The rotations specified for a pattern are cumulative when patterns are nested. The specified rotations are relative to any rotation currently in effect for the pattern.

Remarks: A pattern reference is a drawing statement. Once a pattern has been defined, it can be referenced anywhere in the program that a drawing statement can be specified, except that it cannot be part of the **boundary** specification of a **shape** statement. A pattern definition can contain any number of pattern references, but it cannot reference itself.

The scale factor is calculated by dividing the ratio's numerator by its denominator.

Default Values: If color is not specified, the pattern will be drawn in the color specified in the object definition, or in any other action statement that affects the color of this drawing statement, such as a **make COLOR** statement.

Examples: region Hedge at position 20 20 size 100 100
 rotate 180 pattern Bush

 response to SelectTree
 add to Hedge
 color "red" pattern Tree

 key Several at position 50 50
 blue scale 2:1 pattern Display

See Also: **pattern is** Pattern Definition

pattern is

pattern is Pattern Definition

Purpose: To define a group of graphical drawing statements as a pattern, and identify it for later use in the program.

Model

pattern PATTERN_NAME is

[DRAWING_STATEMENT]...

Where: PATTERN_NAME is the identifier for a pattern.

DRAWING_STATEMENT is a drawing statement.

Remarks: Textual region drawing statements are not permitted in patterns.

A pattern definition can contain any number of pattern references, but it cannot reference itself.

The position at which EASEL OS/2 EE begins to draw a pattern, the *reference point* of the pattern, is the position of the graphics cursor at the time that the pattern is referenced. The coordinates referred to in a pattern definition are in the coordinate system of the graphical object that references the pattern. Thus, it is highly advisable to specify relative movement (e.g., **draw [by]**) rather than absolute movement (e.g., **draw to**) in a pattern definition.

Examples: pattern Logo is
 pattern Star
 move -40 0
 blue draw 100 20
 blue draw -100 0
 blue draw 100 20
 pattern Star

See Also: **pattern** Pattern Reference

plot Action Statement

Purpose: To print EASEL OS/2 EE text and graphical regions on any output device that has a PM driver.

Model

plot { OBJECT_NAME | **screen** } [OPTION_STRING]

Where: OBJECT_NAME is the identifier for the object to be printed.

OPTION_STRING contains one or more of the following:

printer PRINTER_NAME
nobackground
noctext
invertbw

OPTION_
STRING

Options:

printer PRINTER_NAME

Places the job in the printer queue for the selected printer. If more than one queue is assigned to PRINTER_NAME, the default queue is used.
 Default: default printer.

nobackground

Causes all regions to be drawn without any background color. This is most useful for hardcopy devices such as plotters and printers with limited color capability. Some of these devices might output the background in a dark color that could obscure the foreground.
 Default: background is drawn.

noctext

Causes colored textual regions to be drawn as if they are non-colored. Text is drawn in the foreground color of the region.
 Default: colored text is drawn using the background and foreground colors of individual characters.

invertbw

Causes white to be drawn as black and black as white.
 Default: white and black are not inverted.

Use of
Keywords:

... **screen**

Used when you have created a fullscreen interface, and you want to plot all displayed objects.

plot

Remarks: You can print either a selected object or the entire screen.
When an object is plotted, it is drawn on the output page with the same relative position and size as it is displayed on the monitor.

The PM Spooler must be running when the plot statement is executed.

The plot statement will not output image regions, or PM controlled objects such as dialog boxes, action bars, etc.

Examples: response to Key_PtSc
plot screen

response to Print_Key
plot Graph "printer PRINTER1"

Errors: **error in plotting: error parsing plot options**
One of the options in the **plot** action statement is incorrect.

error in plotting: can't find default printer name
A default printer has not been defined. Use the PM Control Panel to define the default printer.

error in plotting: can't find queue and device driver for PRINTER_NAME
No printer queue or device driver is associated with PRINTER_NAME. Use the PM Control Panel to do this.

error in plotting: can't open device context
The PM Spooler is improperly configured.

error in plotting: can't create presentation space
A system error has occurred or there is not enough system memory.

polygon Drawing Statement

Purpose: To draw a polygon in a graphical object.

Model

[**[color]** COLOR] **polygon** XOFFSET YOFFSET[, XOFFSET YOFFSET]

Where: COLOR is a string or integer value representing an EASEL OS/2 EE color.

XOFFSET and YOFFSET are integer values representing the number of coordinate positions to move in the X and Y directions, respectively. The values are expressed in the coordinate system of the object in which the polygon is being drawn.

Use of Keywords: **[color]** COLOR ...
Draws the polygon in the specified color, overriding all other color specifications that otherwise would affect the polygon.

Remarks: A polygon cannot be specified within a **boundary** statement.

All polygons are **solid** (filled), and are closed figures. If you do not specify a closed figure, EASEL OS/2 EE will automatically close the figure by drawing a line from the last coordinates specified back to the first coordinates specified.

Default Values: If color is not specified, the polygon will be drawn in the color specified in the object definition, or in other statements that affect the color of the polygon.

Examples: disabled key at position 250 100
polygon 10 10, 10 -10

position Keyword

Purpose: To position an object in the coordinate system of its parent, or to position a drawing statement in the coordinate system of an object when using the **add to** statement.

Model

at { [position] X Y | **default position** }

Where: X and Y are integer values representing the coordinate position.

Remarks: **at default position** can be used only in an object definition. When the parent is a graphical region, the object is positioned at the graphics cursor of the parent; otherwise, the object is positioned at 0 0.

Examples: key OK at position 50 100

disabled key Stop at position XStart (Height*2)

textual region Bindery window size 30 columns 10 lines
at default position in Panel

<i>See Also:</i>	add to Action Statement	image region Definition
	check box Definition	key Definition
	combination box Definition	list box Definition
	dialog box Definition	multiline entry field Definition
	dialog region Definition	push button Definition
	dropdown list Definition	radio button Definition
	entry field Definition	sense region Definition
	graphical region Definition	static text Definition
	group box Definition	textual region Definition

precision Environment Declaration

Purpose: To specify the precision with which floating point values are displayed.

Model

precision N

Where: N is an integer value representing an integer from 0 to 16 (inclusive).

Remarks: If the precision **0** is specified, the decimal point will be displayed with zero digits to its right.

Default Values: **2** (Truncates the value after two digits to the right of the decimal point.)

Examples: precision 5
precision MaxPrec

See Also: Values

priority is Attribute Definition

Purpose: To define the priority in which an object is drawn, such that it lies beneath or on top of other objects.

Model

priority is PRIORITY_NO

Where: PRIORITY_NO is an integer value representing the priority of the object.

Remarks: The priority specification must appear within the object definition.

When one object is the child of another object, the child is always drawn on top of its ancestor. Otherwise, the priority specification determines which object is on top of the other. An object with a higher priority is drawn on top of an object with a lower priority.

The priority value can be any integer, negative or positive.

You cannot define the priority of the following objects: primary regions, regions with **in desktop** ancestry, dialog boxes, or dialog controls.

Default Values: If no priority is specified, the default value of **0** is used.

Examples: key Box at position 200 100
 priority is 2
 solid box 400 400

disabled key Warning at position 200 100
 priority is 10
 text center "SYSTEM IS GOING DOWN AT 5:00 PM"

See Also: **activate** Action Statement **image region** Definition
 dialog region Definition **key** Definition
 graphical region Definition **sense region** Definition
 Definition **textual region** Definition

priority of Object Inquiry Built-in Function

Purpose: To find and use the current priority value of a specified object.

Model

priority of OBJECT_NAME

Where: OBJECT_NAME is a string value representing the identifier for an object.

Remarks: The value returned is the integer value of the priority associated with the specified object.

Examples: response to line "SYSTEM" from Host
change Warning2 priority to (priority of Warning1)

See Also: **change priority** Action Statement
priority is Attribute Definition

pulldown Definition

Purpose: To define a pulldown item.

Model

```
[ enabled | disabled ] pulldown PD_NAME text PD_TEXT
  [parameter is PAR_STRING]

  [ITEM_DEF_OR_REF] ...
```

end pulldown

Where: PD_NAME is an identifier for the pulldown.

PD_TEXT is the text that will appear in the action bar or pulldown item that displays this pulldown. This text can be any EASEL OS/2 EE string. The text can contain the character ~ to indicate the mnemonic character.

PAR_STRING is a string value representing the parameter of the pulldown.

ITEM_DEF_OR_REF is the definition for, or reference to, a choice, separator, or pulldown.

Use of **text...**
Keywords: Specifies the text associated with the pulldown. This text will be displayed in the action bar or pulldown item that displays this pulldown. If a character in the string is preceded by the ~ character, that character is used as the mnemonic for the pulldown and will be underlined in the text of the pulldown.

Remarks: A pulldown is a list of choices displayed vertically, with optional separators (lines separating the choices).

The parameter of an item is attached to, and is an attribute of, the action bar template or the standalone item template. If the parameter is changed in an action statement, it is changed in the template definition and everywhere it is referenced.

Pulldown items can be defined within an action bar or as stand-alone templates. As stand-alone templates, they can be referenced (and therefore included) in any action bar template.

Cascading pulldowns can be created by nesting pulldowns. You can nest a maximum of 25 levels.

A pulldown definition may not reference itself, directly or indirectly.

Examples:

```

action bar Master_ActionBar is
  pulldown File text "~File"
    choice File_Open text "~Open..."
    disabled choice File_New text "~New"
    separator
    disabled choice File_Save text "~Save"
    disabled choice File_SaveAs text "Save ~As..."
    separator
    choice File_Exit text "E~xit\tF3"
      accelerator is KEY_F3
    end pulldown
  :

# Cascading pulldowns
pulldown Graph text "~Graph"
  pulldown GraphType text "~Type"
    checked choice BarGraph text "~Bar Graph"
    choice PieGraph text "~Pie Graph"
    Choice LineGraph text "~Line Graph"
  end pulldown
  choice GraphCreate text "~Create"
  pulldown GraphErase text "~Erase"
    choice EraseAll text "~All"
    choice EraseCurrent text "Cu~rrent"
  end pulldown
end pulldown

```

See Also: **action bar** is Template Definition
item Reference

push button Definition

Purpose: To define a push button.

Model

```
[ enabled | disabled ] [ visible | invisible ]  
[cancel] [default] push button PB_NAME  
size WIDTH HEIGHT  
at [position] X Y  
in DB_NAME  
[parameter is PAR_STRING]  
[no border]  
[group is GROUP_NAME]  
[text TEXT_STRING]
```

Where: PB_NAME is the identifier for a push button.

WIDTH and HEIGHT are integer values representing the dimensions of the push button in dialog units.

X and Y are integer values representing the position of the push button in dialog units.

DB_NAME is the identifier for the dialog box or dialog region parent object.

PAR_STRING is a string value representing the parameter of the push button.

GROUP_NAME is an identifier for the group that the push button belongs to.

TEXT_STRING is a string value representing the text that will be displayed in the push button.

Use of **cancel**

Keywords: Specifies that this button is to be the cancel push button. This means that if the user presses the Escape key, double-clicks on the system menu, or selects the "Close" choice from the parent's system menu, it causes the response for this button to be taken.

default

Specifies that this button is to be the default push button. This means that if the user presses the Enter key, it causes the response for this button to be taken. The border for this button is thicker than the usual border.

no border

Specifies that the push button is not to have a border.

group is ...

Specifies that this push button is to be grouped with all other dialog control objects that specify this group.

text ...

Specifies the text associated with the push button. This text will be displayed in the center of the push button. If a character in the string is preceded by the ~ character, that character is used as the mnemonic for the button and will be underlined in the text of the button. If the user presses this letter, when not in an entry field, this object will be selected.

Remarks: A push button is a dialog control, and it must have a dialog box or dialog region as its parent.

Only the **push button** keywords, the identifier, the **at [position]** specification, the **in DB_NAME**, and the **size** specification are required in the definition.

When specifying attributes, **parameter**, **no border**, **group is**, and **text** can be specified in any order; all other attributes must be specified in the order shown in the model.

<i>Default</i>	SELECTABILITY:	enabled
<i>Values:</i>	VISIBILITY:	visible
	PARAMETER:	""
	CANCEL:	no
	DEFAULT:	no
	BORDER:	visible
	TEXT:	""

push button

Examples:

- invisible dialog box SaveFile
 - size 160 100 at position 200 200
 - dialog border
 - title bar "Save Output File"
 - system menu
- default push button Save
 - size 38 12 at position 6 4 in SaveFile
 - text "Save"
- cancel push button Cancel
 - size 38 12 at position 56 4 in SaveFile
 - text "Cancel"

radio button Definition

Purpose: To define a radio button.

Model

```
[ enabled | disabled ] [ visible | invisible ] [checked]
radio button RB_NAME
    size WIDTH HEIGHT
    at [position] X Y
    in DB_NAME
    [parameter is PAR_STRING]
    [group is GROUP_NAME]
    [text TEXT_STRING]
```

Where: RB_NAME is the identifier for a radio button.

WIDTH and HEIGHT are integer values representing the dimensions of the radio button in dialog units.

X and Y are integer values representing the position of the radio button in dialog units.

DB_NAME is the identifier for the dialog box or dialog region parent object.

PAR_STRING is a string value representing the parameter of the radio button.

GROUP_NAME is an identifier for the group that the radio button belongs to.

TEXT_STRING is a string value representing the text of the radio button.

Use of **checked**
Keywords: Specifies that this radio button is to be displayed filled in.

group is ...
 Specifies that this radio button is to be grouped with all other dialog control objects using the specified group name. When one radio button in a group is checked, all others in the group are automatically unchecked.

radio button

text ...

Specifies the text associated with the radio button. This text will be displayed to the right of the radio button. If a character in the string is preceded by the ~ character, that character is used as the mnemonic for the radio button and will be underlined in the text of the button. If the user presses this letter, when not in an entry field, this object will be selected.

Remarks: A radio button is a dialog control, and it must have a dialog box or dialog region as its parent.

Only the **radio button** keywords, the identifier, the **at** [**position**] specification, the **in** DB_NAME, and the **size** specification are required in the definition.

When specifying attributes, **parameter**, **group is**, and **text** can be specified in any order; all other attributes must be specified in the order shown in the model.

Default:

SELECTABILITY:	enabled
VISIBILITY:	visible
PARAMETER:	"
CHECKED:	unchecked
GROUP:	none
TEXT:	"

Examples:

```
invisible dialog box SaveFile
  size 160 100 at position 200 200
  dialog border
  title bar "Save Output File"
  system menu

checked radio button HardDisk
  size 48 11 at position 6 20 in SaveFile
  group is Options
  text "to Hard Disk"

radio button Network
  size 48 11 at position 70 20 in SaveFile
  group is Options
  text "to Network"
```

See Also: **is checked** Object/Item Inquiry Built-in Function

read Action Statement

Purpose: To read the contents of (1) an ASCII text file into a textual region or list box, (2) a color-attributed file into a textual region, or (3) a bitmap file into an image region.

Model #1 (for Textual Regions):

```
read [colored] file FILE_NAME into TR_NAME
```

Model #2 (for List Boxes):

```
read file ASCII_FILE_NAME into LB_NAME
```

Model #3 (for Image Regions):

```
read file BMP_FILE_NAME into IR_NAME [using IMAGE_MAP]
```

Where: FILE_NAME is a string value representing the name, including the extension, of an ASCII text file or a color-attributed file.

TR_NAME is the identifier for a textual region.

ASCII_FILE_NAME is a string value representing the name, including the extension, of an ASCII text file.

LB_NAME is the identifier for a list box.

BMP_FILE_NAME is a file containing a bitmapped image. If you do not specify an extension, EASEL OS/2 EE assumes the extension ".bmp". If you do not specify an extension, and there is no file with the extension ".bmp", EASEL OS/2 EE assumes that the file has no extension.

IR_NAME is the identifier for an image region.

IMAGE_MAP is a string value representing the identifier for a previously defined image map, which maps the colors of an image.

*Use of
Keywords:*

read file ...

Reads a file into the textual region, list box or image region. For textual regions, the foreground and background colors of the textual region are those specified in the textual region definition.

read colored file ...

Reads a color-attributed file into the textual region. If the region is a colored textual region, individual characters are colored according to the color attributes stored with each character in the file.

... using ...

Specifies that you want to associate an image map with the image region.

Remarks:

This statement can be used only for textual regions, list boxes, or image regions. Do not use this statement in conjunction with the CText module for the same file.

When the **read** statement is executed for a textual region, the text assumes the font specified in the textual region definition (or the default font for the program), and the text is inserted at the current position of the text cursor. If the text cursor is at a position where there is no text, EASEL OS/2 EE will insert the necessary new blanks up to the position of the text cursor, and then insert the file.

After execution, the text cursor is on the first column of the line immediately following the last line of the read-in file. If there is any text following the text cursor prior to the **read** statement, it is moved to the right and down.

It is highly recommended that you test the value of the built-in function **ioerror** after each **read** statement, to see if the operation was successful.

Valid control characters are **\b** (or **^H**), **\t** (or **^I**), and **\n** (or **^J**). All other control characters contained in the file are ignored. When **\t** is used, it will be expanded to spaces up to the next tab stop.

When the **read** statement is executed for a list box, each line from the text file is inserted according to the sort specification.

If a sort order was not specified for a list box, the contents of the text file are added to the end of the list.

EASEL OS/2 EE does not distinguish between colored and uncolored files when you read files into image regions.

Examples: response to Open
 read file "Report.txt" into Report

 response to LotusBox
 read colored file "Lotus2.col" into ColorMoney

 response to SalesSelect
 read file TodaySales into ViewerA

 # image region
 response to OpenImageFile
 read file "skier" into Display2 using ColorMap2

See Also: **insert** Drawing Statement
 ioerror Action Inquiry Built-in Function
 write Action Statement

record Definition

Purpose: To define a record for use with the File I/O Library subroutines.

Model

```
record RECORD_NAME is
  [ filler FILL_WIDTH
  { VARIABLE_NAME VAR_WIDTH[, PREC] |
    VARIABLE_NAME[ N1[, N2] ] VAR_WIDTH[, PREC] |
    record PD_RECORD_NAME }
  [ filler FILL_WIDTH |
    VARIABLE_NAME VAR_WIDTH[, PREC] |
    VARIABLE_NAME[ N1[, N2] ] VAR_WIDTH[, PREC] |
    record PD_RECORD_NAME ]...
end record
```

Where: RECORD_NAME is the identifier for the record.

FILL_WIDTH is a positive integer constant (greater than zero) denoting the number of columns skipped before the next field begins.

VARIABLE_NAME is either a string, integer, or float variable.

VAR_WIDTH is a positive or negative integer constant specifying the number of columns in the field (including the decimal point and the PRECISION for floats). On input, the sign of the VAR_WIDTH specifier is ignored. On output, the sign of the VAR_WIDTH specifier indicates the justification of the value in the output field (positive = right, negative = left).

PREC (PRECISION) is a positive integer constant specifying the number of digits to the right of the decimal point for a float variable. If no PREC is specified for a float variable in the **record** definition, zero is used. (PREC will be ignored if specified for either string or integer variables.)

VARIABLE_NAME[N1[, N2] ...] is an array variable reference. The array must be a string, integer, or float array, and the reference must be specific (i.e., to a single element in the array). The array can be multidimensional. Subscripts can be either positive integer constants or integer variables.

PD_RECORD_NAME is the name of another record that has been previously defined.

Use of
Keywords: **filler** FILL_WIDTH
 Specifies the number of columns skipped before the next field begins.

Remarks: **filler** cannot be the only specification contained in a record. If you specify **filler** as the last field in a record, it is ignored.

Precision in a record is different from EASEL OS/2 EE precision. EASEL OS/2 EE defaults to a precision of two without a specified precision for floating point. Thus, actions performed on float variables read in by the Input subroutines will contain a decimal point and two digits to the right of the decimal point. If the floating point variable is read in with a precision greater than two specified, but the EASEL OS/2 EE program defaults to a precision of two, the digits after the first two will be truncated.

A record cannot contain itself, either directly or indirectly. The limit on nesting depends on available memory, but is essentially unlimited.

Examples:

```
record Personal_Info is
    Last_Name      15
    First_Name     10
    Address        35
    City           20
    Phone          14
    SS_Number      11
    Married        1
end record

record Company_Info is
    Hiredate       8
    filler          1
    Building       3
    filler          1
    Office         3
    filler          1
    PayGrade       2
end record
```

record

```
record Employee is
  Employee_ID    10
  filler         5
  record Company_Info
  filler         5
  record Personal_Info
end record
```

See Also: File I/O Library (Chapter 2)

refresh Action Statement

Purpose: To redraw a specified object.

Model

refresh OBJECT_NAME

Where: OBJECT_NAME is a string value representing the identifier for an object.

Remarks: The specified object is completely erased and then redrawn.

The **refresh** statement may slow down your program, and therefore should be used only when you find that there is a discrepancy in the screen display.

Example: response to SendSig
refresh object

See Also: **wait** Action Statement

remove

remove Action Statement

Purpose: To remove one or more lines, a text segment, or a text block from a specified textual region, or to remove one or more lines from a list box.

Model #1 (for Textual Regions):

```
remove { [line] LINE_NO [thru [line] LINE_NO] |
         segment [column] COL_NO [line] LINE_NO
           [thru [column] COL_NO [line] LINE_NO] |
         block [column] COL_NO [line] LINE_NO
           [thru [column] COL_NO [line] LINE_NO] }
from TR_NAME
```

Model #2 (for List Boxes):

```
remove [line] LINE_NO [thru [line] LINE_NO] from LB_NAME
```

Where: LINE_NO is an integer value representing the beginning and ending line number of the line, segment, or block in the textual region, or line in the list box.

COL_NO is an integer value representing the beginning and ending column number of the segment or block in the textual region.

TR_NAME is the identifier for a textual region.

LB_NAME is the identifier for a list box.

Use of **remove line ...**
Keywords: Removes the specified line.

... thru line ...
Removes all lines between and including the specified lines.

remove segment ...
Removes all characters from the specified text segment, and any line breaks within the segment.

remove block ...

Removes all characters from the specified text block. Any text to the right of the block is moved to the left, filling in the gap created by its removal. Text never moves up when a block is removed.

Remarks: When text is removed from a textual region with the **remove line** and **remove segment** statements, any text following the removed text is moved to the left, filling the gap created by the removed text.

When a line is removed from a list box, any lines following the removed line are moved up, filling the gap created by the removed line.

You can specify the beginning and ending line, or the beginning and the end of a block or segment, in any order.

Examples: response to Select
remove line 3 thru line 10 from News

response to Select
remove segment column 3 line 1
thru column 3 line 5
from Sales

response to Select
remove block 3 1 thru 3 5 from News

response to Select
remove line 3 from DataArea

See Also: **block** Definition (for Text)
segment Definition

resize Action Statement

Purpose: To increase or decrease an array's range of subscripts.

Model

resize array ARRAY_NAME to DIMENSION

Where: ARRAY_NAME is the identifier for the array.
DIMENSION is an integer value representing the new size of the array.

Remarks: Arrays are resizable in the first dimension only.
If the **resize** statement enlarges the array, data is not disturbed, and additional memory is added. If the **resize** statement decreases the array, some array space will be dropped and the data stored there will be lost.
The following table shows array sizes. If there is not enough memory when enlarging the array with the **resize** statement, use **squeeze memory** to create the additional needed space.

Array Type	Bytes Used
integer	(number of array elements * 4) + 6 + (4 * number of dimensions)
float	(number of array elements * 8) + 6 + (4 * number of dimensions)
string	(number of array elements * 4) + 6 + (4 * number of dimensions) + length of all strings
boolean	(number of array elements + 7) / 8 + 6 + (4 * number of dimensions)

Examples: integer variable ObjectLimit is 10

array size is 30 strings:
string ObjectChildren[1 thru ObjectLimit,3]

```
action RedimensionArray is
  # following statement makes array size
  # 15 elements larger:
  resize array ObjectChildren to (5+ObjectLimit)
  copy (5+ObjectLimit) to ObjectLimit
```

See Also: **array** Definition and Reference
 squeeze memory Action Statement

response to OBJECT Response Definition

Purpose: To respond to the user's selection of an object, or of any object in a class.

Model #1 (for Object Selection)

[**interrupt**] **response to** { OBJECT_NAME | CLASS_NAME }
[ACTION_STATEMENT]...

Model #2 (for Other User-Generated Selections)

[**interrupt**] **response to** { OBJECT_NAME | CLASS_NAME }
on { **activate** | **button1 down** | **button1 double click** | **close** }
[ACTION_STATEMENT]...

[**on** { **activate** | **button1 down** | **button1 double click** | **close** }
[ACTION_STATEMENT]...]...

Where: OBJECT_NAME is the identifier for the object.
CLASS_NAME is the identifier for a class of objects.
ACTION_STATEMENT is an action statement.

Use of **interrupt ...**
Keywords: This allows EASEL OS/2 EE to leave and return to any block currently executing, except a guarded block, when the response is stimulated. EASEL OS/2 EE will first finish any current action statements in the block, then execute the actions in the interrupt response, and finally return to the block and await a stimulus from within the original block.

... on activate
Specifies that the response should be taken if the user activates a region.

... on button1 down

Specifies that the response should be taken if the user clicks on the object with the primary mouse selection button, or selects the object with the keyboard. This is identical to using Model 1.

... on button1 double click

Specifies that the response should be taken if the user double-clicks on the client area of the window with the primary mouse selection button.

... on close

Specifies that the response should be taken if the user selects the "Close" choice from the system menu, or double-clicks on the system menu. A region without a system menu will never receive this event.

Remarks: The response will be invoked only if the specified object is **enabled** at the time that it is selected.

If **response to** CLASS_NAME is used, the selection of any object in the class will stimulate the response.

A response may contain one or more of the **on** clauses in any order. However, a given response may contain no more than one of each type. For example, only one **on close** clause is allowed in any one response.

If there is no **on close** response to invoke for a graphical, image, or textual region, double-clicking on the system menu or selecting the "Close" choice from the system menu destroys the region. If there is no cancel push button response and no **on close** response to invoke for a dialog region, double-clicking on the system menu or selecting the "Close" choice from the system menu destroys the region.

An **on button1 double click** event is always preceded by an **on button1 down** event. The corresponding actions for **on button1 down** (or the generic **response to...** in model 1) will be invoked before the invocation of the actions for **on button1 double click**.

When a **response to...on button1 double click** for a multiple selection list box is invoked, there might not be a currently selected entry in the list box.

*Writing
on activate
Response:*

A region becomes the active region when a user selects it, or one of its children, with the mouse or the keyboard. The active region is displayed with its title bar highlighted.

The **on activate** response statement is invoked when a region becomes the active region as a result of a user action.

response to

If a region is activated due to the execution of an EASEL OS/2 EE statement (such as **activate** or **make...visible**), the **on activate** response for that region is *not* invoked.

The **ReplyToMessage** function (or any function that displays a message box) should not be called from an **on activate** response.

When a region that is not already the active region becomes activated, all of its ancestors are activated as well, beginning with the child and working back, with the furthest removed ancestor being activated last.

An **on activate** response is *not* invoked when a region is first created.

Object/on Clause Combina- tions:

The EASEL OS/2 EE compiler will accept any **on** clause syntax for any response to an object or class name. However, note that some object/**on**-clause combinations are not useful and will never be invoked. The following list describes the combinations that are meaningful.

Object	on Clause	Notes
Graphical region, Image region, Textual region	all	A system menu attribute is required to invoke an on close response.
Sense region, Key	on button1 down on button1 double click	
Dialog box	on activate	Pressing the ESCAPE key, double-clicking on the system menu, or selecting "Close" from the system menu invokes a response to the cancel push button, if there is one.
Dialog region	on activate on close	Pressing the ESCAPE key, double-clicking on the system menu, or selecting "Close" from the system menu invokes a response to the cancel push button, if there is one. If no response to the cancel push button is found, EASEL OS/2 EE looks for and invokes an on close response for that region.
List box	on button1 down on button1 double click	Taken when the user selects an entry.
Push button, Radio button, Check box	on button1 down	

Errors: If OBJECT_NAME or CLASS_NAME are not compile-time values, an error message will be produced.

response to

Examples:

```
response to Command_Key
  enable Cover
  make Prompt visible

interrupt response to Warning
  action WarnUser

response to MainWindow
  on close
    if (ReplyToMessage ("Close","Are you sure?",
      MessageYesNo, 2, MessageNoIcon) = "yes")
    then
      action CloseUp
    end if
  on button1 down
    make MainWindow red
```

response to char and response to line Response Definitions

Purpose: To respond to input from remote or local applications, or from the keyboard.

Model

```
[interrupt] response to { char | line } [MATCH_CLAUSE]...
  from { keyboard | APP_NAME }

[ACTION_STATEMENT]...
```

Where: APP_NAME is an identifier for the application program from which a line of input is received.

ACTION_STATEMENT is an action statement.

MATCH_CLAUSE is a clause describing the input to be matched for invoking the response, following the syntax model:

Model

```
[col NUM1 [thru NUM2] ] { STRING | word |
                           number | blank }
```

Where: NUM1 and NUM2 are compile-time integer values in the range 1-254, representing column positions in the input stream.

STRING is any compile-time string value that must appear in the input. The string must be a literal (in quotation marks) or a string constant. The string must be contiguous.

You can specify as many match clauses as you wish in a **response to char** or **response to line** statement. The response will be invoked when the input received by EASEL OS/2 EE matches every match clause specified for that response. The match clause can specify a particular character string, a blank, any word, any integer number, or any combination of these.

response to char and response to line

Use of Keywords:	interrupt ... This allows EASEL OS/2 EE to leave and return to any block currently executing, except a guarded block, when the response is stimulated. EASEL OS/2 EE will first finish any current action statements in the block, then execute the actions in the interrupt response, and finally return to the block and await a stimulus from within the original block. response to char ... Tells EASEL OS/2 EE that the specified response is to be triggered when specified character(s) are received from the keyboard or from a local or remote application program. The input need not end with a new-line character. In a response to char response statement, the input function returns the characters starting at column 1, up to and including the character that resulted in the match. EASEL OS/2 EE begins further examination at the next column position, which is now called column position 1. When you use response to char response statement in a block, be careful to ensure that EASEL OS/2 EE knows when no more information is expected, so that all available responses—including those outside the block—can be taken. In order to check for a pattern match in an outer block, the pattern in the inner block must not be matched. When you are using response to char, the <i>only</i> way to do this is by using a specific pattern match that excludes any other input. And in order to make sure that all available responses can be taken, use the col specification in the match clause. response to line ... Tells EASEL OS/2 EE that the specified response is to be triggered when a complete line of input is received from the keyboard or from a local or remote application program. EASEL OS/2 EE considers a line to be any stream of characters that ends with a new-line (line feed) character. In a response to line statement, the input function returns the entire line. The next line that EASEL OS/2 EE examines for a match begins with the first column position following the new-line character. ... from keyboard Responds to character input from the keyboard. (See Keyboard Mapping Table)
---------------------	--

*Use of
MATCH_
CLAUSE
Keywords*

- ... **col NUM1 ...**
Specifies exact column position in which the first character of the specified input must appear.
- ... **col NUM1 thru NUM2 ...**
Specifies the range of column numbers in which all the characters in the specified input must appear.
- ... **word**
Specifies that a word must appear in the input. A word is any contiguous group of ASCII characters, including digits but not including blanks, control characters, or delete characters.
- ... **number**
Specifies that a number must appear in the input. A number is a decimal integer, consisting of any digits, optionally preceded by a sign (+ or -) or leading blanks. Embedded blanks, commas, and decimal points are not allowed in numbers. Note that floating point numbers are not taken.
- ... **blank**
Specifies that a blank character must appear in the input.

Remarks:

Specify **response to line** rather than **response to char** whenever possible, to increase processing efficiency.

When a pattern is matched and the response definition is invoked, all characters up to and including the matched pattern are removed from the input stream and copied to the built-in function **input**. Further pattern matching will begin at the next character in the input stream; this character position becomes column 1 of the input stream.

If more than one such response definition is found, the first response definition satisfied is invoked, even if there are other responses earlier in the block that would be satisfied if more of the input were received before responding.

If EASEL OS/2 EE does not find a match between any response definition statement and the current input, even after searching all active blocks, the current input is discarded. If EASEL OS/2 EE does find a match, it performs the action statements specified in the response definition.

Examples:

```
response to line from SalesApp
:
response to char from keyboard
:
```

response to char and response to line

```
response to line col 1 "?" from keyboard
  make Help visible
response to char col 1 "+" from Database
:
response to line
  col 1 ">"
  col 3 thru 8 number
  from Query
```

See Also: Keyboard Mapping Table

response to interval Response Definition

Purpose: To respond to a specified amount of elapsed time.

Model

response to interval SECONDS

[ACTION_STATEMENT]...

Where: SECONDS is an integer value between 1 and 32K inclusive representing the interval, in seconds, before which EASEL OS/2 EE will execute this response.

ACTION_STATEMENT is an action statement.

Remarks: A **response to interval** statement can be specified within a block.

If other actions are taking place while the specified seconds are passing, **response to interval** will take place after those actions are completed. If the actions exceed the specified interval by two or more times, **response to interval** will only execute once, as if only one time interval had elapsed.

Examples: response to interval 60
 read file Status into Status_Region
 change Clock to time

response to interval (2*A)
 :
 :

Errors: **response to interval 0** will produce an error message.

See Also: **response to timeout** Response Definition

response to item Response Definition

Purpose: To respond to the user's selection of an item, or any item in a class.

Model

```
[interrupt] response to item
{ ITEM_NAME [from AB_NAME ] | ITEM_CLASS_NAME }
  [in { REGION_NAME | CLASS_NAME } ]

[ACTION_STATEMENT]...
```

Where: ITEM_NAME is the identifier for a choice or a button.

AB_NAME is the identifier for an action bar template. You must specify the action bar name if more than one template contains the named item. If an item was defined outside of an action bar template, an error will be generated if "from AB_NAME" is referenced.

ITEM_CLASS_NAME is the identifier for an item class containing a group of choices or buttons.

REGION_NAME is the identifier for a region containing the item. If a region is specified, the response will be taken only for the item in that region or class of regions. Otherwise, the response will be taken for all regions that contain the given item.

CLASS_NAME is the identifier for a class of regions.

ACTION_STATEMENT is an action statement.

Use of **interrupt ...**
Keywords: This allows EASEL OS/2 EE to leave and return to any block that is currently executing, except a guarded block, when the response is stimulated. EASEL OS/2 EE will first finish any current action statements from a response in the block, then execute the actions in the interrupt response, and finally return to the block to await a new stimulus from within the original block.

Remarks: A response can be written for any choice in a pulldown or any button in an action bar. A response to a pulldown or a separator will never be invoked.

Buttons do not comply with CUA guidelines.

The response will be invoked only if the specified item is **enabled** at the time that it is selected.

If **response to** ITEM_CLASS_NAME is used, the selection of any choice or pulldown item in the class will stimulate the response.

During the response, the built-in variables **object** and **ancestry** contain, respectively, the name of the item selected and the name of the region, with ancestry, that contains the item.

Examples: interrupt response to item Exit from Primary_ActionBar
in Primary
exit

response to item ItemClass
copy ancestry to Parent
copy text of item object in Parent to
TextOfLastItemSelected

See Also: **ancestry** Response Inquiry Built-in Function
object Response Inquiry Built-in Function

response to line Response Definition

See: **response to char** and **response to line** Response Definitions on page 1-279.

response to low memory Response Definition

Purpose: To respond to minimum free memory threshold.

Model

response to low memory

[ACTION_STATEMENT]...

Where: ACTION_STATEMENT is an action statement.

Remarks: The **response to low memory** statement is taken when EASEL OS/2 EE reaches a minimum threshold of available memory during execution. The threshold is set either by you in the **set low memory threshold** action statement, or by the default value of 5,000 bytes.

Before executing the **response to low memory** statement, EASEL OS/2 EE will attempt to increase available free memory by increasing the program memory size. If this fails, or if the amount of program memory has reached the maximum specified in CONFIG.ESL, the **response to low memory** statement is invoked.

There can be only one **response to low memory** statement in a program. During runtime, EASEL OS/2 EE processes an internal program memory check after execution of each response statement. The **response to low memory** statement is invoked when available memory has fallen below the value specified in the **set low memory threshold** statement, or the default value if none was specified.

The **response to low memory** statement will not be invoked if EASEL OS/2 EE is within a loop, since EASEL OS/2 EE checks memory only after the execution of a response statement, and a loop consists of a series of continuous action statements.

A **response to low memory** statement cannot be inside a block. This response *will* interrupt a guarded block.

You can use the **freesize** function to monitor memory usage within your program. The largest value returned by the **freesize** function is 64K, even when more than this amount of memory is available. Therefore, do not set the low memory threshold above 64K.

response to low memory

Examples: response to low memory
 squeeze memory
 if (freesize < 2000) then
 action Warn_User
 end if

See Also: **freesize** Special Inquiry Built-in Function
 set low memory threshold Action Statement
 squeeze memory Action Statement

response to start Response Definition

Purpose: To respond to program or block initiation.

Model

response to start

[ACTION_STATEMENT] ...

Where: ACTION_STATEMENT is an action statement.

Remarks: The action statement(s) specified in a **response to start** response are executed when the program is started or changed to, or when a block is entered.

Only one **response to start** is allowed in each program, or, if specified in a block, within each block.

Default Values: If there is no **response to start** outside of any block, EASEL OS/2 EE will wait for a stimulus before performing any action.

Examples: response to start
 send Greetings to Recorder

```
response to Send
begin
  response to start
:
end
```

Errors: If there is more than one **response to start** within any block, or outside of all blocks, an error message will be produced.

See Also: **begin** BLOCK Action Statement

response to stimulus Response Definition

Purpose: To allow an external dynamic link library (DLL) to invoke responses in an EASEL OS/2 EE program.

Model

[**interrupt**] **response to stimulus** STIMULUS_NAME
[* | EVENT_FILTER] [* | PARM_FILTER]

[ACTION_STATEMENT]...

Where: STIMULUS_NAME is the identifier for the stimulus.

EVENT_FILTER is an integer value. For the response to be invoked, this value must match the event number provided by the DLL when it invokes the response. If * is specified, any event number will match.

PARM_FILTER is an integer value. For the response to be invoked, this value must match the event parameter provided by the DLL when it invokes the response. If * is specified, any event parameter will match.

ACTION_STATEMENT is an action statement.

Remarks: The stimulus must be declared before it can be referenced in a response.

When searching for a matching response to a stimulus event, EASEL OS/2 EE executes the first response that matches the given event number and parameter. EASEL OS/2 EE searches all currently active blocks for a matching response, until either the outermost block has been searched or a guarded block has been reached. If a matching response is not found, no response is executed and the event is discarded.

Use of **interrupt ...**
Keywords:

This allows EASEL OS/2 EE to leave and return to any block currently executing, except a guarded block, when the response is stimulated. EASEL OS/2 EE will first finish any current action statements in the block, then execute the actions in the interrupt response, and finally return to the block and await a stimulus from within the original block.

*Built-in
Functions:*

Two built-in functions can be used in a **response to stimulus** response definition:

Function	Value Returned
eventnumber	An integer value representing the event number provided by the DLL invoking the response. The meaning of this value is dependent on the DLL.
eventparam	An integer value representing the event parameter provided by the DLL invoking the response. The meaning of this value is dependent on the DLL and the eventnumber.

Examples:

```
# This response will be executed only when
# eventnumber = F1 and eventparam = F2.
response to stimulus StimulusName F1 F2

# This response will be executed whenever
# eventnumber = F1.
response to stimulus StimulusName F1

# This response will be executed whenever
# eventparam = F2.
response to stimulus StimulusName * F2

# This matches any stimulus event.
response to stimulus StimulusName
```

See Also:

stimulus Declaration

response to termination Response Definition

Purpose: To respond to a block being left implicitly.

Model

response to termination

[ACTION_STATEMENT]...

Where: ACTION_STATEMENT is an action statement.

Remarks: This statement can be specified within a block to trigger an action whenever that block is left implicitly. This statement is not taken when the block is explicitly left with the **leave block** keywords.

This statement cannot contain a block, nor can it contain a reference to any action routine or subroutines that contain a block. If it does, an irrecoverable runtime error may occur, and no error message will be generated.

Examples:

```
enabled graphical region Name size 50 20
    at position 0 100 border

disabled invisible key TypeName at position 10 10 in Name
:
enabled graphical region Number size 50 20
    at position 100 100 border

response to Name
    make TypeName visible
    begin
        response to char "\n" from keyboard
            leave block
        response to char from keyboard
            add to TypeName
    text input
        response to termination
            make TypeName invisible
    end
```

response to timeout Response Definition

Purpose: To respond to EASEL OS/2 EE not receiving any stimulus after a specified number of seconds.

Model

response to timeout SECONDS

[ACTION_STATEMENT]...

- Where:* SECONDS is an integer value between 1 and 32K inclusive representing the number of seconds during which EASEL OS/2 EE does not receive any stimulus.
- ACTION_STATEMENT is an action statement.
- Remarks:* The **response to timeout** response statement does not recognize **response to interval** as a stimulus.
- A **response to timeout** statement can be specified within a block.
- There can be only one active **response to timeout** statement at any given time.
- Examples:* response to timeout 60
 stop Remote_Application
 change to program Program_Logo
- See Also:* **response to interval** Response Definition

right [of] Object Inquiry Built-in Function

See: **left [of]** and **right [of]** Object Inquiry Built-in Functions on page 1-206.

save program as Action Statement

Purpose: To save the current state of an EASEL OS/2 EE program, with the changes made during execution, to a new EASEL OS/2 EE binary program file.

Model

save program as PROGRAM_NAME

Where: PROGRAM_NAME is a string value representing the name of the file, including the extension, in which the program is to be stored.

Remarks: The **save program as** statement copies the program, in its current state, to a new program. All objects are saved with their current appearance, attributes, and text and graphics cursor positions; local and global variables are saved with their current values. The old program still exists as it was written, and still continues to run until an **exit** or **change to program** statement is executed. The new program, as amended during execution, has been saved in a new file, identified by the specified value for PROGRAM_NAME.

If you are going to transfer control to this binary program file, be sure that the name includes the **.ebi** extension. If you specify a program name that already exists, the saved program will overwrite the original file. The saved program does not have to be in the current directory; you can specify a full path name for it. When specifying a path, include an *extra* backslash each time one is specified (since a single backslash is the escape character within strings).

If more than 500 objects exist in one program when the **save program as** statement is executed, EASEL OS/2 EE may run out of stack space.

*Global and
Local Values
in a Saved
Program:*

When a program is saved with a **save program as** statement, the value of both global and local variables are saved with it. But if you transfer control to a program that was previously saved, the global variables retain the values they had in the program from which control was transferred.

save program as

Examples:

```
response to End
    save program as "report.ebi"

action SaveMe is
    save program as Name

response to StopSign
    save program as "c:\\cuart-ee\\second.ebi"

response to Small
    add to Design at position 100 150
        box 50 50
    save program as "Display"
    change to program "Show"
```

screen size Environment Declaration

Purpose: To specify the screen coordinate system.

Model

screen size { X Y | **dialog units** | **device units** }

Where: X and Y are integer values representing the number of horizontal and vertical coordinates on the screen. These units are device-independent.

Use of **dialog units**
Keywords: Specifies that the screen coordinate system should be expressed in dialog units. Dialog units are proportional to the size of the PM system font. A dialog unit is defined to be one-quarter of the average character width of the system font in the horizontal direction, and one-eighth of the maximum baseline extent (i.e., the maximum height) of the system font.

device units
 Specifies that the screen coordinate system should correspond to the resolution of the display device. One coordinate in either dimension corresponds to one pixel on the display device.

Remarks: The **screen size** environment declaration should be the first statement in the EASEL OS/2 EE program.

Any object or drawing statement positioned outside of the specified coordinates will not be displayed.

Dialog units are useful for programs that make extensive use of the PM system font. Dialog boxes and controls are always defined in dialog units.

The keywords **dialog units** and **device units** cannot be used with the fullscreen option.

Default If **-fullscreen** is specified at compile time, the default value
Values: of **800 600** is used. Otherwise, the default of **device units** is used.

screen size

Examples: screen size 640 350 # Typical screen size for EGA in
high resolution.

screen size 640 480 # Typical screen size for VGA.

screen size 1024 768 # Typical screen size for 8514.

segment Definition

Purpose: To specify a text segment in a textual region.

Model

segment [**column**] COL_NO [**line**] LINE_NO
[**thru** [**column**] COL_NO [**line**] LINE_NO]

Where: COL_NO is an integer value representing the starting and ending column numbers in the textual region.

LINE_NO is an integer value representing the starting and ending line numbers in the textual region.

Remarks: This specification can be used only with textual regions, in the **insert** drawing statement, **overwrite** drawing statement, **emphasize** action statement, **make segment** action statement, **remove** action statement, or **textual** built-in function. The specification cannot be used for a class of textual regions.

A segment is any set of contiguous characters within a textual region. The starting and ending character positions of a segment may be on the same line or on different lines, and may be specified in either order (first position through last, or last position through first). The segment includes the first and last character positions specified.

Default Values: If the keywords **column** and **line** are not included, the first and third integer values in the definition are used as column values, and the second and fourth integer values in the definition are used as line values.

If you provide only one set of column and line numbers, the segment will be a single character position.

Color attributes are preserved when a segment from a colored textual region is inserted or overwritten into another colored textual region (or another part of the same colored textual region). If the segment is colored but the destination textual region is not, color attributes are dropped.

segment

Examples: response to More
 add to Notes
 insert segment column 3 line 1 thru column 8 line 1
 from Writer

 response to Less
 remove segment 3 1 thru 8 3 from Notes

 response to Change
 add to Notes
 overwrite segment 8 3 thru 3 1 from Writer

See Also: **block** Definition (for Text)
 emphasize Action Statement
 insert Drawing Statement
 make block and **make segment** COLOR Action Statements

overwrite Drawing Statement
 remove Action Statement
 textual Object Inquiry Built-in Function

select and deselect Action Statements

Purpose: To select or deselect a line in a list box.

Model

```
{ select | deselect } line LINE_NO in LB_NAME
```

Where: LINE_NO is an integer value that specifies the line to select or deselect.

LB_NAME is the identifier for a list box.

Remarks: If a line is selected in a list box that is not defined as a multiple selection, the previously selected line will be deselected.

Examples:

```
call ProcessDialogBox1( DidOKCUA )
if DidOKCUA then
    call QueryDialogBox1( SelectedLine )
    deselect line SelectedLine in ControlChoicesList
    if (SelectedLine>1) then
        select line (SelectedLine-1) in ControlChoicesList
    end if

    response to start
    select line 1 in ControlChoicesList
```

See Also: **list box** Definition

selectability of

selectability of Object Inquiry Built-in Function

Purpose: To find the current selectability of an object.

Model

selectability of OBJECT_NAME

Where: OBJECT_NAME is a string value representing the identifier for an object.

Remarks: The value returned by EASEL OS/2 EE is the boolean (**true** or **false**) associated with the selectability of the specified object. If the object is selectable (enabled), the value **true** is returned; if it is not selectable (disabled), the value **false** is returned.

This function is not applicable to items.

Examples: response to SendQuest
if (selectability of Hold) then
:
end if

See Also: **enable** and **disable** Action Statements
enabled and **disabled** Attribute Definitions

selected line Special Inquiry Built-in Function

Purpose: To return the line number of the currently selected line in a list box within a **for each selected line** action statement.

Model

selected line

Examples: action Read_Selected_Reports is
 clear DisplayRegion

 for each selected line of ReportList
 loop
 read file Report_File_Name[selected line] into
DisplayRegion
 end loop

See Also: **for each selected line loop** Action Statement

selected line from

selected line from Object Inquiry Built-in Function

Purpose: To return the line number of the selected line in a list box.

Model

selected line from LB_NAME

Where: LB_NAME is the identifier for a list box.

Remarks: In a single selection list box, the line number of the currently selected line is returned. In a multiple selection list box, only the first selected line will be returned.

Be careful not to confuse **selected line from** with the function **selected line**. If **selected line from** is used in a **for each selected line loop**, only the first selected line will be used every time through the loop, where **selected line** will loop through all selected line numbers.

Examples:

```
response to DoAction
  call ProcessQueryDialogBox(DidOKCUA)
  if (DidOKCUA) then
    copy "Action" selected line from ActionListBox to
    ActionName
    action ActionName
  end if
```

send Action Statement

Purpose: To send information to a local or remote application, or to the errorlog.

Model

send VALUE1 [VALUE2]... **to** { APP_NAME | **errorlog** }

Where: VALUE1 and VALUE2 are any string, integer, float, or boolean values.

APP_NAME is an identifier for the application program name.

Remarks: For the APP_NAME identifier, you specify the same name of the application that you defined in the **application** environment declaration. Remember also that an application must be started with the **start** statement before you can send information to it.

You can send any number of values to an application program or to the errorlog in a single **send** statement. If multiple values are sent, they are concatenated.

When you send data to an application program, you often must end the data with "\r", "\r\n", or "\n". Remote applications usually require "\r", or sometimes "\r\n". Local applications usually require "\n". This simulates the Enter/Return key on the keyboard. Most applications do not recognize input until they receive the line terminator character or character sequence.

You can determine the success or failure of a **send** action statement by testing the **errorlevel** built-in function. A zero value indicates success; any non-zero value indicates failure.

Examples: application Control
start local Control xxx

```
response to line "##" from Host
    send "compute x" "quit_\n" Process to Control
```

```
action Test is
:
    send TestChar to errorlog
```

send

See Also: **application** Environment Declaration
start Action Statement

sense region Definition

Purpose: To define a sense region.

Model

```
[ enabled | disabled ] [ visible | invisible ]
sense region SR_NAME
  size XSIZE YSIZE
  at { [position] X Y | default position }
  [in PARENT_NAME1 [in PARENT_NAME2]... ]
  [window size WINDOW_XSIZE WINDOW_YSIZE
   at [position] WINDOW_X WINDOW_Y]
  [priority is PRIORITY_NO]
  [parameter is PAR_STRING]
```

Where: SR_NAME is the identifier for the sense region.

XSIZE and YSIZE are integer values representing the dimensions of the region's viewport in the parent's coordinate system.

X and Y are integer values representing the position of the region in the parent's coordinate system.

PARENT_NAME1 and PARENT_NAME2 are identifiers for parent objects.

WINDOW_XSIZE and WINDOW_YSIZE are integer values representing the dimensions of the region's window. The window defines how a user selection point is converted into the built-in functions **xcoord** and **ycoord**. (Sense regions have no contents.) Sense regions are used when selection is made by X and Y coordinates rather than through an identified object.

WINDOW_X and WINDOW_Y are integer values representing the position of the region's window.

PRIORITY_NO is an integer value representing the priority of the region.

PAR_STRING is a string value representing the region's parameter.

sense region

Use of **size ...**
Keywords: Defines the size of the region's viewport, and therefore its window if **window size** is not specified, in X Y coordinate.

window size ...
Defines the size of the region's window.

Remarks: A sense region is a rectangular portion of the screen that can be selected. It has no graphical contents or visual appearance. Thus, objects underneath it are visible. A sense region can be used to define a selectable area.

If a sense region is: **The user:**

invisible	cannot select it
enabled and visible	can select it
disabled and visible	cannot select it or objects underneath it

Sense regions cannot have frame styles.

When specifying attributes, the **priority** and **parameter** can be specified in any order.

Only the **region** specification, the identifier, the **at** X Y specification, and the **size** specifications are required. Other attributes are optional, using the defaults shown below.

A sense region cannot have children.

Sense regions cannot be used over PM frame attributes.

In a fullscreen application, a region is disabled by default.

<i>Default</i>	SELECTABILITY:	enabled
<i>Values:</i>	VISIBILITY:	visible
	ANCESTRY:	in primary region
	WINDOW_XSIZE:	XSIZE
	WINDOW_YSIZE:	YSIZE
	WINDOW_X:	0
	WINDOW_Y:	0
	PRIORITY:	0
	PARAMETER:	""

Examples: enabled invisible sense region Select
 size Width Height at position 100 20
 priority is 10

See Also: **enabled** and **disabled** Attribute Definitions
 disabled Attribute Definitions
 parameter is Attribute Definition
 priority is Attribute Definition
 visible and **invisible** Attribute Definitions

separator Definition

Purpose: To define a pulldown separator item.

Model

separator [SEPARATOR_NAME]

Where: SEPARATOR_NAME is an optional identifier for the separator. If you want to reference the separator, you must give it a name.

Remarks: A separator is a horizontal line drawn between pulldown choices. It is used to group choices together. It cannot be moved to or selected.

Separator items can be defined within a pulldown or as a standalone template. As a standalone template, it can be referenced (and therefore included) in any pulldown definition.

Examples:

```
action bar Part_Window_AB is
    pulldown View_Choices text "~View"
        choice View_Part text "~Part"
        separator
        choice View_Description text "~Description"
        choice View_Inventory text "~Inventory"
    end pulldown
end action bar
```

See Also: **action bar** is Template Definition
item Reference
pulldown Definition

set low memory threshold Action Statement

Purpose: To define the minimum free memory needed to run the program.

Model

set low memory threshold to BYTE_SIZE

Where: BYTE_SIZE is an integer value representing the number of bytes needed, up to 64K, to run the program.

Remarks: This statement sets the trigger value for which the **response to low memory** response statement will be executed. If specified, it should be at the beginning of the program. It can be respecified throughout the program. Since EASEL OS/2 EE allocates memory in 64K chunks, the value you set should always be less than 64K.

Default Values: If this statement is not specified, a default value of **5000** bytes is used.

Examples integer WarnValue is 10000
MinValue is 2000

```

response to start
  set low memory threshold to WarnValue
  :
response to low memory
  squeeze memory
  if (freesize <= WarnValue) then
    action ShowWarnMessage
    copy (WarnValue - 5000) to WarnValue
    if (WarnValue < MinValue) then
write TextRegA to file "memreg.sav"
exit
    else
set low memory threshold to WarnValue
    end if
end if

```

See Also: **freesize** Special Inquiry Built-in Function
response to low memory Response Definition
squeeze memory Action Statement

set pointer to

set pointer to Action Statement

Purpose: To change the appearance of the mouse pointer.

Model

set pointer to POINTER_VALUE [**in** DLL_FILE]

Where: POINTER_VALUE is an integer value representing which bitmap pointer to use.

FILE_NAME is the name of a dynamic link library that contains pointer bitmaps.

Remarks: If the "**in** FILE_NAME" is not specified, then the POINTER_VALUE refers to one of the predefined PM pointer bitmaps.

To use the constant names in the **pmptr.inc** file, be sure to include the file. (See "pmptr.inc File" on page 4-20 for a list of the bitmaps.)

Examples: include "pmptr.inc"
 :
 # set pointer to hourglass
 set pointer to SPTR_WAIT

See Also: **pmptr.inc** File (Chapter 4)

shape Drawing Statement

Purpose: To draw one or more closed figures in a graphical object.

Model

```
[solid] [ [color] COLOR] shape
[ [ [color] COLOR] border]
boundary

[MOVE_STATEMENT]
DRAWING_STATEMENT
[ DRAWING_STATEMENT | MOVE_STATEMENT ]...

[boundary

[MOVE_STATEMENT]
DRAWING_STATEMENT
[ DRAWING_STATEMENT | MOVE_STATEMENT ]... ]...
```

end shape

Where: COLOR is an integer or string value representing an EASEL OS/2 EE color.

MOVE_STATEMENT is a **move** or **move arc** statement.

DRAWING_STATEMENT is a **draw**, **draw arc**, **circle**, or **ellipse** statement.

Use of **solid ...**

Keywords: Fills the shape with the shape's color. In the case of multiple boundary specifications in a single **shape** statement, EASEL OS/2 EE will draw all of the boundaries and then fill the shape.

... **[color] COLOR ...**

Draws the shape in the specified color, overriding all other color specifications that otherwise would affect the shape.

[color] COLOR border

Draws the boundaries of the shape in the specified color. The border color specification is separate from the equivalent color specification for the interior of the boundaries.

shape

Remarks: A **shape** statement must contain at least one **boundary** specification. A boundary is a closed figure; each boundary consists of one or more drawing statements that describe the outline of a closed figure. A single **shape** statement can have any number of the **draw**, **draw arc**, **move**, and **move arc** statements. A boundary that contains a **circle** or an **ellipse** is already fully closed, and cannot have additional drawing statements other than the optional **move** or **move arc** statements to position the graphics cursor. A boundary *cannot* contain a **wedge**, **polygon**, or **box** statement.

EASEL OS/2 EE closes any unclosed boundary by drawing a single straight line from the ending coordinate position back to the beginning coordinate position of the boundary.

The first drawing statement in a boundary specification can be a **move** or a **move arc** statement, to position the graphics cursor for use by subsequent statements. If you do use these statements, they must be the first drawing statements in the boundary; it cannot appear later in the boundary, since that would interrupt the continuous lines of the boundary and EASEL OS/2 EE would be unable to make it a closed figure.

If a boundary consists of only **draw** statements, the statements must describe at least two lines of the boundary. If these two lines are parallel, a third line must be specified that is not parallel to the first two.

You can specify any number of boundary specifications between the **shape** and the **end shape** keywords. The boundaries can be specified in any order.

After each boundary is drawn, the graphics cursor is restored to its starting position. Therefore, each boundary that you specify in a single **shape** statement begins at the same position, unless you reposition the graphics cursor by specifying **move** and/or **move arc** statements before the boundary's other drawing statements.

Within a boundary, you can move in an arc, draw an arc of specified degrees, or draw an arc to a specified endpoint. See the **move arc** and **draw arc** statements.

Default Values: If **solid** is not specified, only the shape's perimeter (outline) will be drawn. If no color is specified, the shape will be drawn in the color specified in a pattern reference, object definition, or other statements that contain this drawing statement.

If the **border** (and border color) is omitted, the shape is drawn with the shape's color. If both **solid** and **border** are not specified, the shape is drawn with only a border.

Examples:

```
disabled key Hello at position 25 25
    red shape
    boundary
    draw to 100 0, 120 100, -20 100, 0 0
end shape

key How at position 75 75
    solid green shape
    aqua border
    boundary move to 100 100
    draw 150 0, -150 -100
    draw arc to 100 100 clockwise
    center at 100 50
end shape

key Start at position 10 10
    solid shape
    boundary circle radius 100
    boundary ellipse major 100 minor 40
end shape
```

See Also: **draw arc** Drawing Statement
move Drawing Statement

squeeze memory Action Statement

Purpose: To eliminate fragmentation of memory, thus freeing one larger block of memory for use by the program.

Model

squeeze memory

Remarks: The **squeeze memory** statement is generally used in conjunction with the **response to low memory** response statement and the **freesize** built-in function. This action takes a few seconds to execute, so do not use it unless you must.

EASEL OS/2 EE automatically implements the **squeeze memory** function after you compile your program, and when you specify a **save program** statement.

Examples:

```
action CheckMemory is
  if (freesize >= SizeNeeded) then
    copy true to MemoryOk
  else
    copy false to MemoryOk
    squeeze memory
  end if
```

See Also: **CONFIG.ESL** File (**PRGSIZE** and **GLBSIZE** Options)
(Chapter 3)
freesize Special Inquiry Built-in Function
response to low memory Response Definition
set low memory threshold Action Statement

start Action Statement

Purpose: To execute a local application and start communications with that application, or to start communications with a remote application.

Model

```
start { local LOCAL_APP_NAME PROGRAM_NAME [PARAMETERS] |
       remote REMOTE_APP_NAME PORT [CHARACTERISTICS] }
```

Where: LOCAL_APP_NAME is an identifier for the local application program to be started.

PROGRAM_NAME is a string value identifying the name of the local application program as it is recognized by the operating system.

PARAMETERS is a string value containing one or more parameters to be passed to the local application program.

REMOTE_APP_NAME is an identifier for the remote application program to be started.

PORT is a string value representing the communications port to be used for communication with the remote application program. Note: Parallel ports cannot be used.

CHARACTERISTICS is a string value containing options for the remote terminal characteristics.

Use of ... **local** LOCAL_APP_NAME PROGRAM_NAME ...
Keywords: Starts a local application and communications with that program. You identify the program (LOCAL_APP_NAME) by specifying the same name provided in the **application** statement that defined the application program. You also must specify a string value for the name by which the operating system recognizes the application (PROGRAM_NAME).

... **local** ... PARAMETER
 Specifies a string value that is passed to the local application program as soon as it is started.

... **remote** REMOTE_APP_NAME PORT ...

Starts communications with a remote application. The remote application must already be running before communications can be started. You identify the program (REMOTE_APP_NAME) by specifying the same name provided in the **application** statement that defined the application program. You must also specify one of the following string values for the communications port (PORT) to be used for communication:

"com1"

"com2"

"com3"

... **remote** ... CHARACTERISTICS

Specifies a string value representing the characteristics of the communications port. If more than one characteristic is required, create a string that has a space between each characteristic. The characteristics can appear in any order within the string. The CHARACTERISTICS options and their defaults are:

Characteristic:	Tells EASEL OS/2 EE To:
tandem	Send and obey flow control. (Default.)
-tandem	Not send or obey flow control.
nopar	Not receive or transmit parity. (Default.)
evenpar	Generate even parity.
oddpar	Generate odd parity.
markpar	Generate mark parity (parity bit = 1).
spacepar	Generate space parity (parity bit = 0).
databits 7	Generate 7 bits per character (this is the default, if parity is anything <i>except</i> nopar).
databits 8	Generate 8 bits per character (this is the default, if parity is nopar).
stopbits 1	Generate one stop bit per character. (Default.)

stopbits 2	Generate two stop bits per character.
inbufsize NUM	Set the size of the input buffer used for communications to the specified number of bytes. (Default is 1024.)
outbufsize NUM	Set the size of the output buffer used for communications to the specified number of bytes. (Default is 128.)
xonthresh NUM	Set the XON threshold value. An XON character is sent to the application when the number of unused bytes in the input buffer increases to NUM. (Default is 80% of input buffer size.)
xoffthresh NUM	Set the XOFF threshold value. An XOFF character is sent to the application when the number of unused bytes in the input buffer decreases to NUM. (Default is 20% of input buffer size.)
xonchar CTL_CHAR	Set the control character used for XON flow control. (Default is <code>\^Q</code> .)
xoffchar CTL_CHAR	Set the control character used for XOFF flow control. (Default is <code>\^S</code> .)
timeout SECONDS	Set the maximum amount of time for a connection to be made.
autodial	Send commands to a modem to dial a number and establish a connection.
autoanswer	Automatically wait for and respond to an incoming phone call.
manualdial	Make a connection after the user dials manually (used with an acoustic coupler).

start

manualanswer	Make a connection after the user answers the phone call (used with an acoustic coupler).
direct	Establish a direct connection (used if no modem is attached, or if modem is not supported by CM). (Default.)
number STRING	Dial the specified phone number string (used with AutoDial).
prefix STRING	Dial the specified prefix number string (used with AutoDial).
suffix STRING	Dial the specified suffix number string (used with AutoDial).
pausechar CHAR	Use the character following the keyword as the pause character (used with AutoDial). The default pause character is a comma.

Any one of the following baud rates can be specified:

110	600	4800
150	1200	9600
300	2400	19200

Remarks: You must specify a **start** action statement for all application programs before EASEL OS/2 EE can communicate with these programs.

All applications are global. In one EASEL OS/2 EE program, you might define an application and start it running. In the other programs to which control is transferred, you simply define the application, not restart it. The application continues to run without interruption as control is transferred between programs; the final EASEL OS/2 EE program to use the application may stop it.

You can determine the success or failure of a **start** action statement by testing the value of the **errorlevel** built-in function. A zero value indicates success; any non-zero value indicates failure.

Default Values: The default baud rate is 1200.

If no CHARACTERISTICS entry is specified for a remote application, EASEL OS/2 EE uses the following defaults:

"tandem nopar databits 8 stopbits 1 direct 1200"

Examples: response to start
 start local Prog "DBN52" "1x9"

 response to StartKey
 start remote R "com1" "evenpar"

See Also: **application** Environment Declaration
 stop Action Statement

static text Definition

Purpose: To define a static text object.

Model

```
[ enabled | disabled ] [ visible | invisible ]
static text ST_NAME
  size WIDTH HEIGHT
  at [position] X Y
  in DB_NAME
  [parameter is PAR_STRING]
  [ { left | right | [horizontal] center } [align]
    { top | bottom | vertical center } align ]
  [word wrap]
  [text TEXT_STRING]
```

Where: ST_NAME is the identifier for a static text object.

WIDTH and HEIGHT are integer values representing the dimensions of the object in dialog units.

X and Y are integer values representing the position of the object in dialog units.

DB_NAME is the identifier for the dialog box or dialog region parent object.

PAR_STRING is a string value representing the parameter of the object.

TEXT_STRING is a string value representing the text that will be displayed within the object.

Use of ... **align**
Keywords Specifies the alignment of the text within the object.

word wrap
Specifies that the text in the object will be word wrapped to the next line instead of clipped to the edges of the window.

text ...
Specifies the text that will be displayed within the object.

Remarks: A static text object is a dialog control, and it must have a dialog box or dialog region as its parent.

Only the **static text** keywords, the identifier, the **at** [**position**] specification, the **in** DB_NAME, and the **size** specification are required in the definition.

When specifying attributes, the **parameter**, **align**, **word wrap**, and **text** can be specified in any order; all other attributes must be specified in the order shown in the model.

Word wrap occurs only when the text alignment is top left, top right, or top horizontal center.

Default: SELECTABILITY: **enabled**
 VISIBILITY: **visible**
 PARAMETER: ""
 ALIGNMENT: **top left**
 WORD WRAP: **no**
 TEXT: ""

Examples: invisible dialog box SaveFile
 size 160 100
 at position 200 200
 dialog border
 title bar "Save Output File"
 system menu

static text Instructions
 size 100 8
 at position 0 80 in SaveFile
 horizontal center
 vertical center align
 text "Enter file name and press Save"

stimulus Declaration

Purpose: To specify a logical stimulus name and its associated dynamic link library (DLL) that implements that stimulus.

Model

stimulus STIMULUS_NAME DLL_NAME

Where: STIMULUS_NAME is the identifier for the stimulus.
DLL_NAME is a string value identifying the name of the DLL for this stimulus.

Examples: stimulus APPC_Event "eslappc"
response to stimulus APPC_Event
 if (eventnumber = APPC_ReceiveError) then
 make ErrorWindow visible
 end if

See Also: **response to stimulus** Response Definition

stop Action Statement

Purpose: To stop a local application program and communications with that program, or to stop communications with a remote application program.

Model

```
stop { APPLICATION_NAME | all }
```

Where: APPLICATION_NAME is an identifier for the name of the application program with which communications is to be stopped. This identifier must be the same as that defined in the **application** statement.

Use of Keywords: **stop all**
Stops all local applications and communications with them, and stops communications with all remote applications.

Remarks: A local application that is stopped is terminated immediately, even if there is pending processing for that application. A remote application continues to run; the **stop** statement only terminates communications with that remote application.

If you do not stop a local application or remote application (communications only), EASEL OS/2 EE stops it automatically when an **exit** statement is executed.

Examples: response to Stop
stop Prog

```
action StopApp is
:
stop all
```

See Also: **application** Environment Declaration
start Action Statement

subroutine Declaration

Purpose: To declare an internal EASEL OS/2 EE subroutine or an external subroutine.

Model #1 (for Internal EASEL OS/2 EE Subroutines):

```
subroutine SUB_NAME( [TYPE1:ARG1_NAME[,  
                     TYPE2:ARG2_NAME]... ] )
```

Model #2 (for External Subroutines):

```
subroutine SUB_NAME( [TYPE1:ARG1_NAME[,  
                     TYPE2:ARG2_NAME]... ] )  
library LIB_NAME
```

Where: SUB_NAME is the identifier for the subroutine.

TYPE_n is: **integer**, **string**, **boolean**, or **float**.

ARG_n_NAME is a variable of type TYPE_n.

LIB_NAME is the identifier for an OS/2 dynamic link library.

Remarks: Once you declare an internal EASEL OS/2 EE subroutine, you can reference the subroutine before you define it.

An external subroutine must be declared before it is referenced.

Libraries provided with EASEL OS/2 EE have include files that contain declarations for all the external subroutines in the library.

A subroutine is invoked by referencing the subroutine name, followed by its argument variables enclosed in parentheses, in a **call** action statement.

Example: #1 # EASEL OS/2 EE subroutine declaration
subroutine DrawGraph(string:RegionName,
 string:GraphType)

```
# Subroutine definition. Can appear anywhere in the
# program after the declaration.
subroutine DrawGraph(string:RegionName,
                    string:GraphType) is
    integer A

    copy RegionName to GRegion      # Region used by
                                    # BGraph module
    if (GraphType = "Bar") then
        action GBars
    end if
```

Example: #2: `# External subroutine declaration`
 `subroutine WeekDay(integer:Year,`
 `integer:Month,`
 `integer:Day)`
 `library "datelib"`

Errors: If the type and number of arguments in the EASEL OS/2 EE subroutine declaration and the definition are not the same, a compile-time error is generated when EASEL OS/2 EE reads the definition.

 If there is EASEL OS/2 EE subroutine declaration but no definition, a runtime error is generated when the subroutine is called.

 If an external subroutine is called but never declared, an error is generated at compile time.

See Also: **function** Declaration
 subroutine is Definition
 Chapter 2, Functions and Subroutines
 Chapter 4, Include Files

subroutine is Definition

Purpose: To define a an EASEL OS/2 EE subroutine.

Model

```
subroutine SUB_NAME( [TYPE1:ARG1_NAME[,  
                    TYPE2:ARG2_NAME]... ] ) is  
    [LOCAL_VAR]...  
  
    ACTION_STATEMENT  
    [ACTION_STATEMENT]...
```

Where: SUB_NAME is the identifier for the subroutine.

TYPE_n is: { integer | string | boolean | float }

ARG_n_NAME is a variable of type TYPE_n.

LOCAL_VAR is one or more variable definitions. The variables are local to the subroutine and cannot be referenced outside of it. Local variables are reinitialized whenever the subroutine is called.

ACTION_STATEMENT is an action statement.

Remarks: Arguments of the same type must be declared separately. For example: integer:A, integer:B

Arguments and local variables cannot have the same name.

You cannot define another subroutine or action routine within other subroutines. You can, however, call subroutines, action routines, or functions from within other subroutines, as long as they have been defined or declared previously.

A subroutine is invoked by referencing the subroutine name, followed by its argument variables enclosed in parentheses, in a **call** action statement.

You cannot define constants within a subroutine.

subroutine is

```

Example #1:      subroutine ReadFile(string:InputFile,
                    string:DisplayRegionName) is
                    read file InputFile into DisplayRegionName

response to GetFile
  copy "File.dat" to InputFile
  copy "DisplayRegion" to DisplayRegionName
  call ReadFile(InputFile, DisplayRegionName)

```

```
Example #2:      subroutine Factorial(integer:Number, integer: Result) is
                    integer I
                    J is 1

                    for I= 1 to Number
                    loop
                        copy (J*I) to J
                    end loop
                    copy J to Result

                .

            action GetFactorialOf5 is
                copy 5 to Number
                call Factorial(Number, Result)
                send "Result: "Result"\n" to errorlog
```

See Also: **call** Action Statement
 subroutine Declaration

switch

switch Action Statement

Purpose: To perform pattern matching on a specified string value.

Model

```
switch STRING_VALUE is
  case { char | line } MATCH_CLAUSE is
    [ACTION_STATEMENT]...

  [case { char | line } MATCH_CLAUSE is
    [ACTION_STATEMENT]... ]...

  default is
    [ACTION_STATEMENT]...

end switch
```

Where: STRING_VALUE is a string variable.

ACTION STATEMENT is an action statement.

MATCH_CLAUSE is a clause describing the input to be matched for invoking the case, following the syntax model:

Model

```
[col NUM1 [thru NUM2] ] { STRING | word |
                           number | blank }
```

Where: NUM1 and NUM2 are compile-time integer values in the range 1-254, representing column positions in the input stream.

STRING is any compile-time string value that must appear in the input. The string must be a literal (in quotation marks) or a string constant. The string must be contiguous.

<i>Use of MATCH_ CLAUSE Keywords</i>	... col NUM1 ... Specifies exact column position in which the first character of the specified input must appear. ... col NUM1 thru NUM2 ... Specifies the range of column numbers in which all the characters in the specified input must appear. ... word Specifies that a word must appear in the input. A word is any contiguous group of ASCII characters, including digits but not including blanks, control characters, or delete characters. ... number Specifies that a number must appear in the input. A number is a decimal integer, consisting of any digits, optionally preceded by a sign (+ or -) or leading blanks. Embedded blanks, commas, and decimal points are not allowed in numbers. Note that floating point numbers are not taken. ... blank Specifies that a blank character must appear in the input.
<i>Remarks:</i>	Only the first case that matches the pattern will be executed. If no case matches, the default action statements will be executed. The switch statement is an alternative to using nested if/then/else statements.
<i>Examples:</i>	<pre>switch (Appl_Var) is case char "bad transmission" is : case char "enter name" is : case char "more data" is : default is : end switch</pre>
<i>Errors:</i>	The default case is required, and it must be the last case specified. It does not need an action statement, but it cannot be omitted.

text Drawing Statement

Purpose: To display text in a graphical region, key, dialog control, or item.

Model #1 (for Graphical Regions and Keys):

```
[ [color] COLOR] [font FONT_NAME]
[text [ align | center ] ] TEXT_STRING
```

Model #2 (for Dialog Controls and Items):

text TEXT_STRING

Where: COLOR is an integer or string value representing an EASEL OS/2 EE color.

FONT_NAME is a string value representing an EASEL OS/2 EE or PM character font.

TEXT_STRING is a string value representing the text to be displayed.

Use of Keywords

... **[color]** COLOR ...
Specifies the color in which the text string is to be displayed.

... **font** FONT_NAME
Specifies the font in which the text string is to be displayed.

... **text align** ...
Left-justifies text at the current position of the text cursor. This is the default value of the **text** statement.

... **text center** ...
Centers text both horizontally and vertically at the current position of the text cursor.

- Remarks:** The **text** drawing statement cannot be used in image regions, textual regions, sense regions, dialog boxes, list boxes, or separators.
- Any characters, including Extended ASCII characters (if the font is non-proportional PM), can be specified in the text to be displayed.
- For graphical regions and keys, each **text** statement is independent of all others; any graphical region or key can have any number of different text statements.
- For graphical regions and keys, after the text has been displayed, the text cursor is moved to the position immediately following the string. The graphics cursor does not move.
- Any PM font can be displayed in a graphical region.
- Dialog controls and items do not have a graphics cursor.
- Default Values:** If color or font is not specified, the text string will be drawn in the color or font currently in effect.
- If you do not specify **center** or **align**, the text string defaults to **align**.
- Examples:**
- ```
key OK at position 10 10
 "OK"

key GoAhead at position 100 100
 blue font "medium" text align "Press any key"
 blue font Normal_Font text align "to continue"

disabled key Agenda at position 30 45
 color COLOR_A text center Agenda

response to Send
 add to Buttons
 move to 25 25
 "Button " Button1
 # Displays text string consisting of
 # string literal "Button ", with
 # contents of string variable
 # Button1 appended.
```
- See Also:** **font** Reference  
**font is** Font Definition  
**overwrite** Drawing Statement

---

## **text of Object Inquiry Built-in Function**

*Purpose:* To find the text string associated with an object.

*Model*

---

**text of** OBJECT\_NAME

---

*Where:* OBJECT\_NAME is a string value representing the identifier for an object. The object cannot be a key.

*Remarks:* The value returned is the text string associated with the specified object. For regions and dialog boxes, this is the text displayed within the title bar, if one is present. For dialog controls, this is the text displayed within the control.

*Examples:* response to OKinDialogBox  
copy text of NameEntryField to EmployName

*See Also:* **text** Drawing Statement

---

## text of item Item Inquiry Built-in Function

*Purpose:* To inquire about the text associated with an item.

*Model*

---

**text of item** ITEM\_NAME [**from** AB\_NAME] **in** REGION\_NAME

---

*Where:* ITEM\_NAME is a string value representing the identifier for an item.

AB\_NAME is a string value representing the identifier for an action bar template. You must specify the action bar name if more than one template contains the named item.

REGION\_NAME is a string value representing the identifier for a region containing the item.

*Remarks:* The value returned is the text associated with the specified item.

*Example:* response to item ItemClass  
copy ancestry to Parent  
copy text of item object in Parent to  
TextOfLastItemSelected

*See Also:* **text** Drawing Statement

---

## **textual Object Inquiry Built-in Function**

**Purpose:** Returns the contents of (1) a specified line, segment, or block of a textual region, or (2) a specified line of a list box.

*Model #1 (for Textual Regions):*

---

```
textual { line LINE_NO |
 segment [column] COL_NO [line] LINE_NO
 [thru [column] COL_NO [line] LINE_NO] |
 block [column] COL_NO [line] LINE_NO
 [thru [column] COL_NO [line] LINE_NO] } from TR_NAME
```

---

*Model #2 (for List Boxes):*

---

```
textual line LINE_NO from LB_NAME
```

---

**Where:** LINE\_NO is an integer value representing the line number in the textual region or list box, or the beginning and ending line numbers of the segment or block in the textual region.

COL\_NO is an integer value representing the beginning and ending column numbers of the segment or block in the textual region.

TR\_NAME is the identifier for a textual region.

LB\_NAME is the identifier for a list box.

**Use of** **textual line ...**  
**Keywords:** Returns the contents of the specified line. If the line does not exist, the null string is returned and an error message is generated.

**textual segment ...**  
Returns a string containing all characters within the specified segment. Line breaks are converted to new-line characters in the string. All color attributes are dropped.

**textual block ...**

Returns a string containing all lines from the specified block, each line separated by a single blank character. All color attributes are dropped.

**Remarks:** If **thru** and an ending column and line number are not specified, the segment or block will contain only one character.

**Examples:**    response to GetLine  
                  copy textual line 1 from Manual to CurrentList  
  
                  response to CopyBox  
                  copy textual block column 10 line 4  
                  thru column 50 line 8 from OldText to NewString

**See Also:**    **block** Definition (for Text)  
                  **segment** Definition

---

## textual region Definition

*Purpose:* To define a textual region, which is a rectangular portion of the screen in which text can be displayed.

*Model*

---

```
[enabled | disabled] [visible | invisible] [[color] COLOR]
[primary] [colored] textual region TR_NAME
{ size XSIZE YSIZE | window size COLUMNS columns LINES lines }
at { [position] X Y | default position }
 [in PARENT_NAME1 [in PARENT_NAME2]...]
 [[color] COLOR foreground]
 [priority is PRIORITY_NO]
 [parameter is PAR_STRING]
 [[size | [color] COLOR] border]
 [title bar TITLE_STRING]
 [system menu]
 [horizontal scroll bar [scroll by SCROLL_INC]]
 [vertical scroll bar [scroll by SCROLL_INC]]
 [action bar AB_NAME]
 [minimize button [using MINIMIZE_ICON_FILE]]
 [maximize button]
 [pointer [is] POINTER_FILE]
 [font FONT_NAME]

[DRAWING_STATEMENT]...
```

---

*Where:* COLOR is a string or integer value representing a color.

TR\_NAME is the identifier for the textual region.

XSIZE and YSIZE are integer values representing the dimensions of the textual region's viewport in the parent's coordinate system. These refer to the dimensions of the viewport itself, and do not include the space occupied by the region's frame.

COLUMNS and LINES are integer values representing the number of columns and lines of the window, in the textual region's coordinate system.

X and Y are integer values representing the position of the textual region's viewport in the parent's coordinate system (not the position of the outside frame).



PARENT\_NAME1 and PARENT\_NAME2 are identifiers for parent objects. If it is a primary region, you cannot specify ancestry; it is always **desktop**. To place a region outside of a primary region, specify its ancestry as **in desktop**.

PRIORITY\_NO is an integer value representing the priority of the region. The primary region and regions whose ancestry is desktop cannot be assigned a priority.

PAR\_STRING is a string value representing the region's parameter.

TITLE\_STRING is a string value representing the title that will appear in the textual region's title bar.

SCROLL\_INC is an integer value representing the number of columns (if horizontal scroll) or number of lines (if vertical scroll) by which the textual region's window position will be changed when one of the arrows in the scroll bar is pressed.

AB\_NAME is a string value representing the identifier for an action bar template.

MINIMIZE\_ICON\_FILE is a string value representing the name of the file, including the extension, containing the icon to use when the textual region is minimized.

POINTER\_FILE is a string value representing the name of the file, including the extension, containing the bit map to use as a pointer whenever the pointer is over this textual region.

FONT\_NAME is a string value representing the font to be used in the region. The font cannot be proportionally-spaced, outline (scaleable), bold, italic, underline, or strikeout.

DRAWING\_STATEMENT is a drawing statement. This cannot be a graphical drawing statement.

*Use of  
Keywords:*

**primary**

Specifies that this textual region is the program's primary region.

**color ...**

Specifies that this textual region is a colored textual region, which can have background and foreground colors specified for individual characters.

**size ...**

Defines the size of the textual region's viewport in X Y coordinates. The window size in columns and lines is inferred from the viewport size and the font being used.

## textual region

### **window size ...**

Defines the size of the textual region's window in columns and lines. The viewport size in X Y coordinates is inferred from the window size.

### **... border**

Specifies the type of border present around the textual region. If **border** or **COLOR border** is specified, a one-pixel border of the color indicated (white, by default) will be drawn. If **size border** is specified, the textual region will be surrounded by a sizing border that can be used to resize the region. If other frame attributes are specified for the region, then the **border** specification will cause a one pixel wide border in the PM border color to be displayed.

### **title bar ...**

specifies that the region will contain a title bar. This will generate a title bar that can be used for repositioning the region. The title bar will contain the text specified.

### **system menu**

Specifies that the textual region will contain a system menu. The system menu button will be in the upper left corner.

### **horizontal scroll bar ...**

Specifies that the textual region will contain a horizontal scroll bar. The scroll bar will be located along the bottom edge of the textual region just inside the border, if one exists. The textual region's contents will be scrolled as specified by **SCROLL\_INC**.

### **vertical scroll bar ...**

Specifies that the textual region will contain a vertical scroll bar. The scroll bar will be located along the right-hand edge of the textual region just inside the border, if one exists. The textual region's contents will be scrolled as specified by **SCROLL\_INC**.

### **action bar ...**

Specifies that the region will contain an action bar.

### **minimize button ...**

Specifies that the textual region will contain a minimize button in the upper right corner.

### **maximize button**

Specifies that the region will contain a maximize button in the upper right corner. (The maximize button will turn into a restore button when the textual region is maximized).

**pointer is ...**

Specifies the file containing the bit map to use as a pointer.

**font ...**

Specifies the font to be used.

*Remarks:*

In a windowed application, the primary region must be the first object you define. There can be only one primary region.

Only the **textual region** specification, the identifier, the **at [position]** specification, and the **size** specifications are required, and they must be specified in the order shown in the model. Other attributes are optional, using the default values shown below.

Any of the attributes after ancestry can be specified in any order.

A textual region cannot be the ancestor of any object.

When you define a textual region's size or window size, you are actually defining two characteristics: the region's viewport and the region's window.

The region's viewport and window are always the same size. There is no scaling.

The region size only includes the area of the viewport itself. It does not include the frame attributes, which are external to the viewport. If the size of the primary region is the same as the screen size, you will not be able to see any frame attributes.

Textual regions without frame attributes (such as title bar, scroll bar, etc.) will not display partially clipped lines and columns.

Double-clicking on the system menu or selecting the "Close" choice from the system menu destroys the region, if there is no **on close** response to invoke for the region.

Disabling a region disables its viewport and its frame.

For colored textual regions, the color specifications in the definition are for the background and foreground of the entire textual region. The color specifications for individual characters in a colored textual region override the color specifications in the definitions. Colors for individual characters are specified with either the **make block** or **make segment** action statement, or with the CText module.

Specifying **size** causes the number of columns and lines to be determined based on font size.

**textual region**

|                |                          |                           |
|----------------|--------------------------|---------------------------|
| <i>Default</i> | SELECTABILITY:           | <b>enabled</b>            |
| <i>Values:</i> | VISIBILITY:              | <b>visible</b>            |
|                | COLOR (background):      | <b>black</b>              |
|                | ANCESTRY (primary):      | <b>in desktop</b>         |
|                | ANCESTRY:                | <b>in primary region</b>  |
|                | COLOR (foreground):      | <b>white</b>              |
|                | PRIORITY:                | <b>0</b>                  |
|                | BORDER:                  | <b>invisible</b>          |
|                | COLOR (border):          | <b>white</b>              |
|                | SCROLL_INC (horizontal): | <b>one column</b>         |
|                | SCROLL_INC (vertical):   | <b>one line</b>           |
|                | FONT:                    | <b>medium<sup>1</sup></b> |

*Examples:*    enabled visible color 26 textual region SecondaryWindow1  
                 size 260 160  
                 at position 140 95  
                 in desktop  
                 color 27 foreground  
                 size border  
                 title bar "Text Window"  
                 system menu  
                 horizontal scroll bar scroll by 6  
                 vertical scroll bar scroll by 16  
                 action bar SecondaryWindow1AB  
                 minimize button using "TEXT.ICO"  
                 maximize button  
                 pointer is "TEXT.PTR"

In the example above, color 26 picks up the PM window background color, and color 27 picks up the PM window foreground color.

|                  |                                    |                                     |
|------------------|------------------------------------|-------------------------------------|
| <i>See Also:</i> | <b>border</b> Attribute Definition | <b>in</b> Keyword                   |
|                  | <b>color</b> Keyword               | <b>make block</b> and <b>make</b>   |
|                  | <b>enabled</b> and <b>disabled</b> | <b>segment</b> COLOR Action         |
|                  | Attribute Definitions              | Statements                          |
|                  | <b>font</b> Reference              | <b>parameter is</b> Attribute       |
|                  | <b>font is</b> Font Definition     | Definition                          |
|                  |                                    | <b>priority is</b> Attribute        |
|                  |                                    | Definition                          |
|                  |                                    | <b>visible</b> and <b>invisible</b> |
|                  |                                    | Attribute Definitions               |

<sup>1</sup> This is the font used if none is specified as an environment declaration.

---

## time Special Inquiry Built-in Function

*Purpose:* To determine the current time.

*Model*

---

**time**

---

*Remarks:* The value returned for the **time** function is the current 24-hour time as HH:MM:SS, where:

HH is the hour, and appears as 1 or 2 digits

MM is the minute, and appears as 2 digits

SS is the second, and appears as 2 digits

To get the time in the format specified in the PM Control Panel or CONFIG.SYS COUNTRY setting, use the **LocalTime( )** function.

*Examples:* response to interval 1  
change Ticker to  
font "medium" time

*See Also:* **date** Special Inquiry Built-in Function  
**LocalTime( )** Function (Chapter 2)

---

top [of] and bottom [of] Object Inquiry Built-in Functions

*Purpose:* To find and use the current top and bottom coordinate positions describing the smallest rectangular area occupied by an object, its contents, children, and children's contents.

*Model*

---

{ top | bottom } [of] OBJECT\_NAME

---

*Where:* OBJECT\_NAME is a string value representing the identifier for an object.

*Remarks:* The values returned are based upon the specified object's contents (drawing statements), children, and children's contents, and are expressed in the coordinates of the object. If the object has no contents, EASEL OS/2 EE returns values of zero for these functions.

Avoid using the **top [of]** or **bottom [of]** functions for a scaled graphical region that contains text, or in a program whose defined screen size is different than the display resolution. Since text does not scale, the **top [of]** and **bottom [of]** functions can return incorrect values in these situations.

For a graphical object and dialog region, the following values are returned:

- |                    |                                                                     |
|--------------------|---------------------------------------------------------------------|
| <b>top [of]</b>    | The topmost Y coordinate occupied by the contents of the object.    |
| <b>bottom [of]</b> | The bottommost Y coordinate occupied by the contents of the object. |

For an image region, the following values are returned:

- |                    |                                                                                        |
|--------------------|----------------------------------------------------------------------------------------|
| <b>top [of]</b>    | The topmost Y coordinate occupied by the image file currently displayed in the region. |
| <b>bottom [of]</b> | Always 0.                                                                              |

For a textual region, the following values are returned:

- |                 |                                                                        |
|-----------------|------------------------------------------------------------------------|
| <b>top [of]</b> | Always 1. This is line number of the first line in the textual region. |
|-----------------|------------------------------------------------------------------------|

**bottom [of]**                      The line number of the last line of text in the textual region.

For a list box, the following values are returned:

**top [of]**                        Always 1. This is line number of the first line in the list box.

**bottom [of]**                    The line number of the last line of text in the list box.

*Examples:*                      response to Move  
                                      if ((bottom of Zoom) < MaxZ)  
                                         then add to Zoom  
                                             move to Width ((bottom of Zoom) + 10)

*See Also:*                      **left [of]** and **right [of]** Object Inquiry Built-in Functions  
                                      **xmiddle [of]** and **ymiddle [of]** Object Inquiry Built-in Functions

## turn trace

---

### turn trace Action Statement

*Purpose:* To record, in the errorlog, every action statement that EASEL OS/2 EE performs. Used for debugging.

*Model*

---

**[turn] trace { on | off }**

---

*Remarks:* Typically, you trace only those areas of the EASEL OS/2 EE program that are untested or that are not producing the results you want, because EASEL OS/2 EE operates much more slowly when tracing is on.

EASEL OS/2 EE begins to record its actions as soon as it encounters a **turn trace on** statement, and ceases tracing as soon as it encounters a **turn trace off** statement.

*Examples:* response to NewShade  
turn trace on  
copy "red" to Color  
make Box1 color Color  
turn trace off

*See Also:* **EASEL.CMD** OS/2 Command File (Chapter 3)



---

## Type Conversions

**Remarks:** Type conversions are never performed inside an expression. They are performed only on values in EASEL OS/2 EE statements. The following conversions are performed:

| <b>Value</b>   | <b>Converts to...</b>                                                    |
|----------------|--------------------------------------------------------------------------|
| <b>boolean</b> | <b>integer, float, string</b>                                            |
| <b>string</b>  | <b>integer, float</b> , Identifier (name)<br>(with and without ancestry) |
| <b>float</b>   | <b>integer, string</b>                                                   |
| <b>integer</b> | <b>float, string</b>                                                     |
| Identifier     | <b>string</b> (names)<br>(with and without ancestry)                     |

When a string contains blanks or alphabetic or special characters, EASEL OS/2 EE starts at the beginning of the string and proceeds according to the following rules, for conversions of string values to integer or floating point values:

1. It ignores any leading blanks.
2. It examines the first nonblank character(s) in the string. If the string consists of digits (optionally preceded by a minus sign), EASEL OS/2 EE interprets the value as being an integer. If the string consists of digits and a decimal point, the digits before the decimal point and after the decimal point up to the first non-digit are regarded as a floating point value.
3. It terminates at the first character that is not a digit or decimal point. If it terminates before locating a number, the value is zero.

For conversion of boolean values to integer or floating point values, the boolean value **true** is converted to the value **1**; **false** is converted to the value **0**.

A floating point value that is converted to an integer value is truncated at the decimal point; the value will not be rounded.

## Type Conversions

For conversions of identifiers with ancestry to string values and vice versa, an identifier in the form of:

`CHILD in PARENT1 in ... in PARENTn`

has an equivalent string value in the form of:

`"PARENTn/.../PARENT1/CHILD"`

*Errors:* If you specify a value that cannot be converted to the required type, an error message will be generated.

---

## **uncheck** Action Statement

See: **check** and **uncheck** Action Statements on page 1-64.

---

## Values

*Purpose:* In every EASEL OS/2 EE statement in which a value is required, you must specify a value of a particular type, or use the default value.

| Category                                       | Types                                                             | Default                                              | Examples                                                                      |
|------------------------------------------------|-------------------------------------------------------------------|------------------------------------------------------|-------------------------------------------------------------------------------|
| Literal                                        | <b>string</b><br><b>integer</b><br><b>float</b><br><b>boolean</b> |                                                      | "status OK"<br>5<br>27.96<br>false                                            |
| Constant                                       | <b>string</b><br><b>integer</b><br><b>float</b><br><b>boolean</b> | <b>""</b><br><b>0</b><br><b>0.00</b><br><b>false</b> | CompanyName<br>Counter_Max<br>Pi<br>Yes                                       |
| Variable                                       | <b>string</b><br><b>integer</b><br><b>float</b><br><b>boolean</b> | <b>""</b><br><b>0</b><br><b>0.00</b><br><b>false</b> | Password<br>EmployeeNumber<br>Grand_Total<br>FirstTime                        |
| Variable<br>preceded by<br>a unary<br>operator | <b>integer</b><br><b>float</b><br><b>boolean</b>                  |                                                      | -A<br>-Profit<br>not AlreadyDone                                              |
| Parenthesized<br>Expression                    | <b>string</b><br><b>integer</b><br><b>float</b><br><b>boolean</b> |                                                      | (Message = "Final step")<br>(A * B + C)<br>(A * 1.5)<br>((A > B) and (C > D)) |
| Built-in<br>Function                           | <b>string</b><br><b>integer</b><br><b>boolean</b>                 |                                                      | members of Class2<br>xsize of GraphArea<br>ioerror                            |

**Remarks:** Built-in functions do not return floating point values.

Every value you specify must match the type required in the particular syntax model. Throughout this publication, syntax models indicate the type of value required in each statement. If you specify a value of the wrong type, EASEL OS/2 EE converts the value if it can (see “Type Conversions” on page 1-347).

The keywords **true** and **false** can be used wherever a boolean value can be used.

**See Also:** Built-in Functions                      **precision** Environment  
                 **constant** Definition                      Declaration  
                 Expressions, Operators,                      Type Conversions  
                 and Operands                      **variable** Definition

---

## variable Definition

*Purpose:* A variable is a named entity that holds a data value that can be changed during execution. Once defined, it can be referenced in the program (or in other programs to which control has been transferred).

*Model*

---

```
[global] { integer [variable] | float [variable] |
 boolean [variable] | string [variable] |
 [string] variable }
VAR1 [is VALUE1]
[VAR2 [is VALUE2]]...
```

---

*Where:* VAR1 and VAR2 are identifiers representing variable names.

VALUE1 and VALUE2 are compile-time values of the same type as their associated variables, representing the initial values of the variables.

*Use of* **global ...**  
*Keywords:* Specifies a **global** variable. When one of the programs transfers control to another program, the values of the global variables are retained and are available for use by the new program. All programs that use a global variable must define it as a global variable, specifying the same name and the same type. If a program executed with the **change to program** action statement initializes the global variable, the initialization does not override the value left behind by any earlier program.

*Remarks:* The **variable** statement defines one or more variables. A variable must be defined before it is referenced.

When you define a variable, you specify the name, type, and optional initial values of one or more variables. The type that you specify for a variable determines the type of value required for the initialization. You do not need to initialize variables when you define them.

*Default* integer: **0**  
*Values:* float: **0.00**  
 boolean: **false**  
 string: **""** (null string)

If you specify **string** or **variable** alone, EASEL OS/2 EE assumes **string variable**.

If the variable is not defined as **global**, the variable defaults to a local variable. In a local variable, the value assigned will be retained only while that program is executing.

*Examples:* integer variable MaxHeight is 10000  
  
 global variable MessageCount  
  
 boolean SuccessFlag is true  
  
 variable City is "BOSTON"  
                   State is "MA"  
                   Country is "USA"

|                  |                                           |                                                    |
|------------------|-------------------------------------------|----------------------------------------------------|
| <i>See Also:</i> | <b>append</b> Action Statement            | <b>copy</b> Action Statement                       |
|                  | <b>array</b> Definition and Reference     | <b>extract from</b> Action Statement               |
|                  | <b>change to program</b> Action Statement | <b>length of</b> Special Inquiry Built-in Function |
|                  |                                           |                                                    |

---

## visibility of Object Inquiry Built-in Function

*Purpose:* To find out the current visibility of an object.

*Model*

---

### visibility of OBJECT\_NAME

---

- Where:* OBJECT\_NAME is a string value representing the identifier for an object.
- Remarks:* The value returned is the boolean value (**true** or **false**) associated with the visibility of the specified object. If the object is visible, the value **true** is returned; if it is invisible, the value **false** is returned.
- Example:*
- ```
response to GoButton
  if (visibility of PleaseWait) then
    action MustWait
  else
    action GoAhead
  end if
```
- See Also:* **make invisible** and **make visible** Action Statements
visible and **invisible** Attribute Definitions

visible and invisible Attribute Definitions

Purpose: To define an object as visible or invisible.

Model

{ **visible** | **invisible** }

Remarks: The visibility specification must appear within the object definition.

If you define an object as **invisible**, it is not drawn. If you define an object as **visible**, it can be seen except under the following conditions:

- if any of its ancestors is invisible.
- if its position is outside the boundaries of the screen.
- if it is the child of a graphical or image region, and it is outside the boundaries of the window size specified for the region.

Making a modal dialog box visible disables all other windows in the application until the dialog box is made invisible or deleted.

Care should be taken when changing the visibility of dialog control objects. Note that unpredictable results may occur.

If an object is invisible, it cannot be selected even if it is enabled.

Default Values: All objects are visible by default.

Examples: visible key List at position 20 50

invisible textual region Holder size 20 20
at position 100 100

visible and invisible

See Also: **button** Definition
 check box Definition
 choice Definition
 combination box
 Definition
 dialog box Definition
 dialog region Definition
 dropdown list Definition
 entry field Definition
 graphical region
 Definition
 group box Definition
 image region Definition

key Definition
list box Definition
make invisible and **make visible** Action Statements
multiline entry field
Definition
pulldown Definition
push button Definition
radio button Definition
sense region Definition
static text Definition
textual region Definition
visibility of Object Inquiry
Built-in Function

wait Action Statement

Purpose: To suspend all EASEL OS/2 EE processing for a specified number of seconds.

Model

wait SECONDS

Where: SECONDS is an integer value representing the number of seconds that EASEL OS/2 EE processing is to be suspended. The maximum value is determined by the SLEEPMAX parameter in the CONFIG.ESL file. This value defaults to 60.

Remarks: The range of integer values allowed for the SECONDS value is 0 to SLEEPMAX. The time that EASEL OS/2 EE waits is approximate. The actual elapsed time might be longer than the time you specify.

When a **wait** statement is executed, EASEL OS/2 EE queues any stimuli that it might receive. After the specified number of seconds has elapsed, EASEL OS/2 EE processes any queued stimuli in the order in which they were received.

If **wait 0** is specified, EASEL OS/2 EE will update (redraw) the screen immediately, even if all the action statements in the response have not finished executing. (Usually, EASEL OS/2 EE updates the screen only when the *last* action statement in a response definition has been executed.)

Examples:

```

response to line from Database
    make Chart visible
    wait 10

action HoldForApp is
    wait WaitTime

response to Go
    make object green
    wait 0
    make object red
    wait 0
    make object green
  
```

wait

See Also: **CONFIG.ESL** File (Chapter 3)
 refresh Action Statement

wedge Drawing Statement

Purpose: To draw a wedge in a graphical object, such as a pie chart.

Model

```
[solid] [ [color] COLOR] wedge [radius] RADIUS
from DEG1 degrees to DEG2 degrees
```

Where: COLOR is an integer or string value representing an EASEL OS/2 EE color.

RADIUS is an integer value representing the radius of the wedge.

DEG1 and DEG2 are integer values representing the absolute starting and stopping angles (degrees) of the wedge, respectively. The values are expressed in the coordinate system of the object in which the wedge is being drawn.

Use of **solid ...**
Keywords: Fills the wedge with the wedge's color.

[color] COLOR ...
 Draws the wedge in the specified color, overriding all other color specifications that otherwise would affect the wedge.

Remarks: A wedge cannot be specified within a **boundary** statement.
 EASEL OS/2 EE draws the wedge, starting at the current position of the graphics cursor. (This is the center of the circle of which the wedge is a part.) The final position of the graphics cursor is the same as its initial position, and the text cursor is moved to that position.

Default If **solid** is not specified, just the wedge's perimeter
Values: (outline) will be drawn.

If color is not specified, the wedge will be drawn in the color specified in a pattern reference, object definition, or other statements that would affect the wedge.

wedge

Examples: key Steady at position 10 10
 wedge radius 100 from 5 degrees to 90 degrees

 disabled key Timer at position 50 75
 solid color 5 wedge radius 30
 from 3 degrees to 40 degrees

while loop Action Statement

Purpose: To specify a set of action statements to be performed repeatedly, as long as a specified condition is true.

Model

```
while ( BOOLEAN_EXPRESSION ) loop [LOOP_NAME]

    [ACTION_STATEMENT]...
    [leave loop [LOOP_NAME] ]

end loop [LOOP_NAME]
```

Where: **BOOLEAN_EXPRESSION** is a boolean value.
 LOOP_NAME is the identifier for the loop.
 ACTION_STATEMENT is an action statement.

Use of **leave loop**
Keywords: The **leave loop** action statement can be one of the action statements in a **while loop** statement. Upon execution, EASEL OS/2 EE will exit from the loop before the condition specified in the **while** statement has failed, and will continue execution at the statement following the **end loop** keywords. If you specify a **leave loop** statement, you must specify it within the definition of the loop itself. You cannot specify it in an action routine that is called from inside the loop.

leave loop LOOP_NAME

Specifies a name in the **leave loop** statement. This name might identify the loop in which the **leave loop** is specified, or it might identify an outer loop. If you specify an outer loop, EASEL OS/2 EE leaves the specified outer loop and all inner loops. If you do not specify a name in the **leave loop** statement, EASEL OS/2 EE exits from the innermost loop that contains the **leave loop** statement, even if the innermost loop has a name.

while loop

Remarks: The **while loop** statement can be specified within a response definition or an action routine definition. It can be specified within a conditional action statement or within other loops, to any number of levels. You can specify blocks in loops, and vice versa. If you specify a block within a loop, execution of a **leave loop** statement causes all blocks contained in the loop to become inactive.

You specify both the condition and the actions in the **while loop** statement.

The **while loop** statement must end with the **end loop** keywords. The **leave loop** statement is optional.

EASEL OS/2 EE first evaluates the **BOOLEAN_EXPRESSION** specified in the **while** clause. If the boolean value is **true**, EASEL OS/2 EE performs the action statements specified in the loop. When EASEL OS/2 EE reaches the **end loop** keywords, it evaluates the boolean once again. If the result is **true**, EASEL OS/2 EE returns to the top of the loop and performs the action statements again. This cycle continues for as long as the boolean value is **true**.

If the boolean value is **false**, during either the initial evaluation or a later evaluation, EASEL OS/2 EE ignores the loop and continues execution at the action statement (if any) immediately following the **end loop** statement. If the boolean is initially **false**, the loop will not be executed at all.

The identifier used to name the loop must be unique within the EASEL OS/2 EE program; there must be nothing else in the program with that name. Specifying a name in the **while loop** statement means that you *must* include a name in the **end loop** statement which exactly matches the name specified in **while loop**.

Examples:

```
copy 1 to I
while (I <= 10) loop
    add to Axes
    move 20 0
    pattern Hashmark
    copy (I + 1) to I
end loop
```



```
response to Help2
  while (Int <= 5) loop
    action Action_Sub
    if (Action_flag != "OK") then
      leave loop
    end if
    copy (Int + 1) to Int
  end loop
  copy Action_flag to Flag_log
```

See Also: **for loop** Action Statement
 if Conditional Action Statement
 leave loop Action Statement

window xposition [of] and **window yposition [of]** Object Inquiry Built-in Functions

Purpose: To find and use (1) the origin coordinates of a graphical, image, sense, textual, or dialog region's window, or (2) the line number of the first line displayed in a list box.

Model

{ **window xposition** | **window yposition** } [of] OBJECT_NAME

Where: OBJECT_NAME is a string value representing the identifier for a region or list box.

Remarks: The values returned are expressed in the coordinates of the specified object.

For a graphical, image, sense, or dialog region, the values returned are:

window xposition [of] The X coordinate of the origin of the window (expressed in dialog units for dialog regions).

window yposition [of] The Y coordinate of the origin of the window (expressed in dialog units for dialog regions).

For a textual region, the values returned are:

window xposition [of] Column number of the leftmost character in the window.

window yposition [of] Line number of the topmost line in the window.

For a list box, the values returned are:

window xposition [of] Always 1.

window yposition [of] Line number of the first line displayed in the list box.

window xposition [of] and window yposition [of]

Examples: response to PanLeft
 change Book window position by (-StepSize) 0
 if ((window xposition of Book) <= (left of Book)) then
 :
 end if
 if ((window xposition of Book) +
 (window xsize of Book) <
 (right of Book)) then
 make PanRight visible
 end if

Errors: If you use these functions with any other type of object,
EASEL OS/2 EE returns values of zero, along with an
appropriate error message.

window xsize [of] and **window ysize [of]** Object Inquiry
Built-in Functions

Purpose: To find and use the current size of a graphical, image, sense, textual, or dialog region's window.

Model

{ **window xsize** | **window ysize** } [of] REG_NAME

Where: REG_NAME is a string value representing the identifier for a region.

Remarks: The **window xsize [of]** and **window ysize [of]** functions can be applied to regions only. The values returned are expressed in the coordinates of the specified region.

For a graphical, image, sense, or dialog region, the values returned are:

window xsize [of] The X dimension of the size of the region's window (expressed in dialog units for dialog regions).

window ysize [of] The Y dimension of the size of the region's window (expressed in dialog units for dialog regions).

For a textual region, the values returned are:

window xsize [of] The number of columns in the window.

window ysize [of] The number of lines in the window.

Examples: response to StretchHeight
 change Graph window size by 0 (-Squish)
 if ((window ysize of Graph) <= Squish)
 then make StretchHeight invisible
 end if

Errors: If you use these functions with any other type of object, EASEL OS/2 EE returns values of zero, along with an appropriate error message.

window yposition [of] Object Inquiry Built-in Function

See: **window xposition [of]** and **window yposition [of]** Object Inquiry Built-in Functions on page 1-364.

window ysize [of]

window ysize [of] Object Inquiry Built-in Function

See: **window xsize [of]** and **window ysize [of]** Object Inquiry
Built-in Functions on page 1-366.

write Action Statement

Purpose: To write (1) a textual region to an ASCII text file or a color-attributed file, or (2) an image region or graphical region to a file.

Model #1 (for Textual Regions):

```
write [ append | destructive ] TR_NAME to [colored] file FILE_NAME
```

Model #2 (for Image Regions and Graphical Regions):

```
write REGION_NAME to file FILE_NAME
```

Where: TR_NAME is the identifier for a textual region.

FILE_NAME is a string value representing the name of the file, including the extension, to which the textual region is to be copied.

REGION_NAME is the identifier for an image region or graphical region.

Use of **write ...**

Keywords: Writes the contents of the region to the specified file. For textual regions, if the file already exists (and **destructive** is not specified), the operation fails and the function **ioerror** is set to **true**. For image and graphical regions, **write** is always destructive, replacing the contents of any existing file.

write append ...
 Appends the contents of the region to the end of the contents of the specified file. If no file with the specified name exists, a new file will be created.

write destructive ...
 Replaces the contents of an existing file with the contents of the textual region. If no file with the specified name exists, a new file will be created.

... to file FILE_NAME
 Writes a region to a file. If a colored textual region is written to an ASCII file, the color specifications for individual characters will be dropped.

write

... to colored file FILE_NAME

Writes a textual region to a color-attributed file. If an uncolored textual region is written to a colored file, the file will contain color specifications based upon the background and foreground colors of the textual region. If a colored textual region is written to a colored file, the file will retain the color specifications for individual characters.

Remarks: None of the **write** statements affect the region in any way.

EASEL OS/2 EE does not distinguish between colored and uncolored files when you write files from image regions or graphical regions. The file is in PM standard bitmap format and is a pseudo-color file. If the image region is color mapped, the current color map is used.

You should test the value of the built-in function **ioerror** after each **write** statement, to see if the operation was successful.

Examples: response to char "OK" from keyboard
write Manual to file "Archive.dat"

response to OverwriteFile
write destructive FirstDraft to file Store

action AddText is
write append Chapter to colored file Book

if (ioerror) then
action WarnUser
end if

Errors: If for any reason the operating system will not let EASEL OS/2 EE create, append to, or delete the specified file, the built-in function **ioerror** will be set to **true**, and an error message will be generated.

See Also: **insert** Drawing Statement
ioerror Action Inquiry Built-in Function
read Action Statement

xcoord and ycoord Response Inquiry Built-in Functions

Purpose: To find and use the coordinates of the pointer's position when the current object was selected.

Model

{ xcoord | ycoord }

Remarks: The **xcoord** and **ycoord** functions can be specified only within a **response to OBJECT** response definition.

For graphical objects and image regions, the values returned are:

xcoord The X coordinate of the point selected, in the coordinate system of the object for which a response is taken.

ycoord The Y coordinate of the point selected, in the coordinate system of the object for which a response is taken.

For textual regions, the values returned are:

xcoord The column number of the character selected.

ycoord The line number of the character selected.

Errors: If you specify these functions within any response other than a **response to OBJECT**, EASEL OS/2 EE will return **0** as the value of the function. If you specify these functions within a response to any other type of object, 0 will be returned.

Examples: response to BlueKey
 move to xcoord ycoord
 :

xcursor [of] and ycursor [of] Object Inquiry Built-in Functions

Purpose: To find and use the coordinates of the current position of the graphics cursor in a specified graphical object, or the position of the text cursor in a specified textual region.

Model

{ xcursor | ycursor } [of] OBJECT_NAME

Where: OBJECT_NAME is a string value representing the identifier for an object.

Remarks: The values are expressed in the object's own coordinates. For graphical objects, the values returned are:

xcursor [of] The X coordinate of the graphics cursor position.

ycursor [of] The Y coordinate of the graphics cursor position.

For textual regions, the values returned are:

xcursor [of] The column number of the text cursor position.

ycursor [of] The line number of the text cursor position.

Examples: response to MoveMenu
copy (xcursor of Z) to SaveX
copy (ycursor of Z) to SaveY
add to Z
pattern Q
move to SaveX SaveY # Restore original graphics
cursor position
:

Errors: If you specify these functions within a response to any other type of object, 0 will be returned.

xmiddle [of] and ymiddle [of] Object Inquiry Built-in Functions

Purpose: To find and use the X and Y coordinates corresponding to the middle of the specified object's contents.

Model

{ **xmiddle** | **ymiddle** } [of] OBJECT_NAME

Where: OBJECT_NAME is a string value representing the identifier for an object.

Remarks: The values returned are based upon the values of the **top**, **bottom**, **left**, and **right** functions, and are expressed in the coordinates of the object. If the object has no contents, EASEL OS/2 EE returns values of zero for these functions. The values returned are calculated as:

xmiddle [of] = (left + right) / 2

ymiddle [of] = (top + bottom) / 2

For graphical objects, these functions return X and Y coordinates. For textual regions, these functions return column and line numbers.

Avoid using the **xmiddle [of]** or **ymiddle [of]** functions for a scaled graphical region that contains text, or in a program whose defined screen size is different than the display resolution. Since text does not scale, the **xmiddle [of]** and **ymiddle [of]** functions can return incorrect values in these situations.

Examples: response to More
 add to Box
 move to (xmiddle of Box) (ymiddle of Box)
 circle radius 1

See Also: **top [of]** and **bottom [of]** Object Inquiry Built-in Functions
left [of] and **right [of]** Object Inquiry Built-in Functions

xposition [of] and yposition[of] Object Inquiry Built-in Functions

Purpose: To find and use the origin coordinates of a specified object.

Model

{ xposition | yposition } [of] OBJECT_NAME

Where: OBJECT_NAME is a string value representing the identifier for an object.

Remarks: The values returned are the X and Y coordinates of the origin of the specified object, and are expressed in the coordinates of the object's parent. For regions, this is the coordinates of the object's viewport.

Examples: response to MoveZoomLeft
 if ((xposition of Zoom) > StepSize) then
 change Zoom position by (-StepSize) 0
 end if

xsize [of] and ysize [of] Object Inquiry Built-in Functions

Purpose: To find and use the current size of a region.

Model

{ xsize | ysize } [of] REG_NAME

Where: REG_NAME is a string value representing the identifier for a region.

Remarks: The values returned are the X and Y size of the specified region, expressed in the coordinates of the object's parent.

Examples: response to MoveChapRight
if (((xposition of Chapter) + (xsize of Chapter))
 <= (ScreenWidth - StepSize)) then
 :
end if

ycoord Response Inquiry Built-in Function

See: **xcoord** and **ycoord** Response Inquiry Built-in Functions on page 1-371.

ycursor [of] Object Inquiry Built-in Function

See: **xcursor [of]** and **ycursor [of]** Object Inquiry Built-in Functions on page 1-372.

ymiddle [of] Object Inquiry Built-in Function

See: **xmiddle [of]** and **ymiddle [of]** Object Inquiry Built-in Functions on page 1-373

yposition [of] Object Inquiry Built-in Function

See: **xposition [of]** and **yposition[of]** Object Inquiry Built-in Functions on page 1-374.

ysize [of]

ysize [of] Object Inquiry Built-in Function

See: **xsize [of]** and **ysize [of]** Object Inquiry Built-in Functions on page 1-375.

Chapter 2. Functions and Subroutines

Overview

This chapter describes the syntax for referencing the functions and subroutines provided with the IBM EASEL Version 1.1 for OS/2 Extended Edition (EASEL OS/2 EE) Libraries.

The following describes the libraries provided with EASEL OS/2 EE.

Date Library

The Date Library provides functions for performing date operations and validation.

The Date Library contains the following functions:

DaySpan()
LocalDate()
LocalTime()
Validdate()
WeekDay()

EASEL OS/2 EE Library

The EASEL OS/2 EE library provides functions for performing fullscreen color value inquiry and update operations and active region inquiries.

The EASEL OS/2 EE Library contains the following functions:

Es1QueryEgaColor()
Es1QueryFocus()
Es1SetEgaColor()

File I/O Library

The File I/O Library provides subroutines for performing read and write operations with files.

The File I/O Library contains the following subroutines:

CloseFile()	ReadRecordAtLine()
GetError()	SetBufferSize()
OpenFile()	SetIndentSize()
ReadLineNumber()	SetTabSize()
ReadNext()	WriteString()
ReadNextRecord()	WriteRecord()

Math Library

The Math Library provides functions and subroutines for performing mathematical and statistical operations.

The Math Library contains the following functions:

ABS()	FLOOR()
ARCOS()	LN()
ARCSIN()	LOG()
ARCTAN()	MOD()
ARCTAN2()	PI()
CEIL()	POWER()
COS()	SIN()
E()	SQRT()
EXP()	TAN()

The Math Library contains the following subroutines:

MAX()	STDDEV()
MAXINT()	STDDEVINT()
MEAN()	SUM()
MEANINT()	SUMINT()
MEDIAN()	VARIANCE()
MEDIANINT()	VARIANCEINT()
MIN()	
MININT()	

Message Library

The Message Library provides a function for displaying a message box.

The Message Library contains the following function:

ReplyToMessage()

ABS()

ABS() Function

Action: Returns the absolute value of X.

Model

ABS(X)

Where: X is a float value.

Return Type: **float**

Include File: **mathlib.inc**

Remarks: The include file, which contains the function declaration, must be referenced using the **include** statement before the function is invoked.

A function is invoked by referencing it wherever a value of the same type as the return type can be used.

See Also: Math Library
include Reference (Chapter 1)

ARCCOS() Function

Action: Returns the angle in degrees whose cosine is X.

Model

ARCCOS(X)

Where: X is a float value.

Return Type: **float**

Include File: **mathlib**

Remarks: The include file, which contains the function declaration, must be referenced using the **include** statement before the function is invoked.

A function is invoked by referencing it wherever a value of the same type as the return type can be used.

Errors: X must be in the range $-1.0 \leq X \leq 1.0$, or the function returns the value -1 and places the following message in the EASEL OS/2 EE errorlog:

ARCCOS: argument < -1 or > 1; returning -1.

See Also: Math Library
include Reference (Chapter 1)

ARCSIN() Function

Action: Returns the angle in degrees whose sine is X.

Model

ARCSIN(X)

Where: X is a float value.

Return Type: **float**

Include File: **mathlib.inc**

Remarks: The include file, which contains the function declaration, must be referenced using the **include** statement before the function is invoked.

A function is invoked by referencing it wherever a value of the same type as the return type can be used.

Errors: X must be in the range $-1.0 \leq X \leq 1.0$, or the function returns the value -1 and places the following message in the EASEL OS/2 EE errorlog:

ARCSIN: argument < -1 or > 1; returning -1.

See Also: Math Library
include Reference (Chapter 1)

ARCTAN() Function

Action: Returns the angle in degrees whose tangent is X.

Model

ARCTAN(X)

Where: X is a float value.

Return Type: **float**

Include File: **mathlib.inc**

Remarks: The include file, which contains the function declaration, must be referenced using the **include** statement before the function is invoked.

A function is invoked by referencing it wherever a value of the same type as the return type can be used.

ARCTAN returns a value in the range -90 to +90 degrees.

See Also: Math Library
include Reference (Chapter 1)

ARCTAN2() Function

Action: Returns the angle in degrees whose tangent is Y/X.

Model

ARCTAN2(Y, X)

Where: Y and X are float values.

Return Type: **float**

Include File: **mathlib.inc**

Remarks: The include file, which contains the function declaration, must be referenced using the **include** statement before the function is invoked.

A function is invoked by referencing it wherever a value of the same type as the return type can be used.

Note: Y must be the first argument specified.

Errors: X cannot be 0.0, or the function returns the value -1.0 and places the following message in the EASEL OS/2 EE errorlog:

ARCTAN2: second argument = 0; returning -1.0.

See Also: Math Library
include Reference (Chapter 1)

CEIL() Function

Action: Returns the smallest integer larger than or equal to X.

Model

CEIL(X)

Where: X is a float value.

Return Type: **integer**

Include File: **mathlib.inc**

Remarks: The include file, which contains the function declaration, must be referenced using the **include** statement before the function is invoked.

A function is invoked by referencing it wherever a value of the same type as the return type can be used.

Errors: X must be in the range $-2147483648 \leq X \leq 2147483647$, or the function returns the value 0 and places one of the following messages in the EASEL OS/2 EE errorlog:

CEIL: argument > largest integer; returning 0.

CEIL: argument < smallest integer; returning 0.

See Also: Math Library
include Reference (Chapter 1)

CloseFile()

CloseFile() Subroutine

Action: Performs the necessary overhead required when closing a file.

Model

CloseFile(FILE_ID)

Return Type: FILE_ID is a user-defined name that follows the rules for an EASEL OS/2 EE identifier. It must be declared previously by the **fileid** statement.

Include File: **fileio.inc**

Remarks: The include file, which contains the subroutine declaration, must be referenced using the **include** statement before the subroutine is invoked.

A subroutine is invoked by referencing it in a **call** action statement.

You can determine the success or failure of the subroutine by testing the value of the **errorlevel** built-in function. A zero value indicates success; any nonzero value indicates failure.

<i>Errors:</i>	errorlevel	Error Message/Explanation
	1	Error locking global memory. Indicates a serious problem. Contact your support representative.
	2	File has not been opened. This error occurs when you try to close a file that has not been opened.
	8	File handle is invalid. Contact your support representative.

See Also: File I/O Library
call Action Statement (Chapter 1)
errorlevel Action Inquiry Built-in Function (Chapter 1)
fileid Definition (Chapter 1)
include Reference (Chapter 1)

COS() Function

Action: Returns the cosine of the argument in degrees.

Model

COS(X)

Where: X is a float value.

Return Type: **float**

Include File: **mathlib.inc**

Remarks: The include file, which contains the function declaration, must be referenced using the **include** statement before the function is invoked.

A function is invoked by referencing it wherever a value of the same type as the return type can be used.

To convert degrees to radians, multiply degrees by 0.0174532926.

See Also: Math Library
include Reference (Chapter 1)

DaySpan()

DaySpan() Function

Action: Returns a positive integer that is the number of days in the range between two dates.

Model

DaySpan(YEAR1, MONTH1, DAY1, YEAR2, MONTH2, DAY2)

Where: YEAR1 and YEAR2 are integer values representing the full years of the specified dates, i.e., 1989, 1990, etc. YEAR1 specifies the year of the earlier date in the range.

MONTH1 and MONTH2 are integer values representing the months of the specified dates, i.e., 2 for February, 3 for March, etc. MONTH1 specifies the month of the earlier date in the range.

DAY1 and DAY2 are integer values representing the days of the specified dates. DAY1 specifies the day of the earlier date in the range.

Return Type: **integer**

Include File: **datelib.inc**

Remarks: The include file, which contains the function declaration, must be referenced using the **include** statement before the function is invoked.

A function is invoked by referencing it wherever a value of the same type as the return type can be used.

If an invalid date is specified, -1 is returned.

Example: copy (DaySpan(1989, 6, 5, 1990, 8, 3)) to Num_Of_Days

See Also: Date Library
include Reference (Chapter 1)

E() Function

Action: Returns the constant value of e, which is 2.71828...

Model

E()

Return Type: **float**

Include File: **mathlib.inc**

Remarks: The include file, which contains the function declaration, must be referenced using the **include** statement before the function is invoked.

A function is invoked by referencing it wherever a value of the same type as the return type can be used.

See Also: Math Library
include Reference (Chapter 1)

EslQueryEgaColor() Function

Action: In fullscreen applications, this function returns the EGA or VGA color value for color numbers 18 through 25.

Model

EslQueryEgaColor(ESL_COLOR_NUMBER)

Where: ESL_COLOR_NUMBER is an integer value.

Return Type: **integer**

Include File: **esllib.inc**

Remarks: The include file, which contains the function declaration, must be referenced using the **include** statement before the function is invoked.

A function is invoked by referencing it wherever a value of the same type as the return type can be used.

This function is provided for use with fullscreen programs only. It will not work with EASEL OS/2 EE windowed programs.

See Also: EASEL OS/2 EE Library
include Reference (Chapter 1)

EslQueryFocus() Function

Action: Returns the name of the EASEL OS/2 EE region that currently has the keyboard input focus.

Model

EslQueryFocus()

Return Type: **string**

Include File: **esllib.inc**

Remarks: The include file, which contains the function declaration, must be referenced using the **include** statement before the function is invoked.

A function is invoked by referencing it wherever a value of the same type as the return type can be used.

This function gives the EASEL OS/2 EE program a way to determine where keyboard input should be echoed when there are multiple regions on screen.

Do not call this function in a response to an object, as this may yield unpredictable results.

See Also: EASEL OS/2 EE Library
include Reference (Chapter 1)

EslSetColor()

EslSetColor() Function

Action: In fullscreen applications, this function sets the EGA or VGA color values for color numbers 18 through 25.

Model

EslSetColor(ESL_COLOR_NUMBER, EGA_VGA_NUMBER)

Where: ESL_COLOR_NUMBER is an integer value representing an EASEL OS/2 EE color number.
EGA_VGA_NUMBER is an integer value representing an EGA/VGA color value.

Return Type: **integer**

Include File: **esllib.inc**

Remarks: The include file, which contains the function declaration, must be referenced using the **include** statement before the function is invoked.

A function is invoked by referencing it wherever a value of the same type as the return type can be used.

This function is provided for use with fullscreen programs only. It will not work with EASEL OS/2 EE windowed programs.

If the display adapter is not an EGA or VGA, the function returns 0; otherwise, it returns a nonzero value.

<i>Default EGA/VGA Color Set:</i>	Color Number	EGA/VGA Color Value
	18	7 (Light Gray)
	19	56 (Dark Gray)
	20	11 (Light Blue)
	21	5 (Dark Purple)
	22	37 (Bright Pink)
	23	52 (Orange)
	24	38 (Gold)
	25	20 (Brown)

<i>EGA/VGA Color Values by Color Group:</i>	Color Group	EGA/VGA Color Values
	Black	0
	White	63
	Reds	4, 32, 36
	Greens	2, 6, 10, 14, 16, 18, 19, 22, 23, 26, 30, 34, 42, 48, 50, 51, 58
	Blues	1, 8, 9, 11, 13, 15, 17, 25, 29, 33, 41, 43, 49, 57
	Yellows	54, 55, 62
	Purples	5, 12, 21, 40, 45, 47, 61
	Oranges	38, 52
	Cyans	24, 27, 31, 49
	Pinks	39, 53, 37
	Peaches	44, 46, 60
	Grays	7, 56
	Blue-grays	3, 35
	Brown	20
	Tan	28

See Also: EASEL OS/2 EE Library
include Reference (Chapter 1)

EXP()

EXP() Function

Action: Returns the exponential function of X (the value which is e to the X power).

Model

EXP(X)

Where: X is a float value.

Return Type: **float**

Include File: **mathlib.inc**

Remarks: The include file, which contains the function declaration, must be referenced using the **include** statement before the function is invoked.

A function is invoked by referencing it wherever a value of the same type as the return type can be used.

See Also: Math Library
include Reference (Chapter 1)

FLOOR() Function

Action: Returns the largest integer smaller than or equal to X.

Model

FLOOR(X)

Where: X is an integer value.

Return Type: **integer**

Include File: **mathlib.inc**

Remarks: The include file, which contains the function declaration, must be referenced using the **include** statement before the function is invoked.

A function is invoked by referencing it wherever a value of the same type as the return type can be used.

Errors: X must be in the range $-2147483648 \leq X \leq 2147483647$, or the function returns the value 0 and places one of the following messages in the EASEL OS/2 EE errorlog:

FLOOR: argument > largest integer; returning 0.

FLOOR: argument < smallest integer; returning 0.

See Also: Math Library
include Reference (Chapter 1)

GetError() Subroutine

Action: Sets a string variable to the error message associated with the error level passed to it.

Model

call GetError(ERROR_LEVEL, MESSAGE)

Where: ERROR_LEVEL is an integer variable that specifies an error code. To set ERROR_LEVEL, copy the value of the built-in function **errorlevel** to it.

MESSAGE is a string variable that contains a short error message after the call to **GetError()** is made.

Include File: **fileio.inc**

Remarks: The include file, which contains the subroutine declaration, must be referenced using the **include** statement before the subroutine is invoked.

A subroutine is invoked by referencing it in a **call** action statement.

You can determine the success or failure of the subroutine by testing the value of the **errorlevel** built-in function. A zero value indicates success; any nonzero value indicates failure.

See Also: File I/O Library
call Action Statement (Chapter 1)
errorlevel Action Inquiry Built-in Function (Chapter 1)
include Reference (Chapter 1)

LN() Function

Action: Returns the value of the natural logarithm.

Model

LN(X)

Where: X is a float value.

Return Type: **float**

Include File: **mathlib.inc**

Remarks: The include file, which contains the function declaration, must be referenced using the **include** statement before the function is invoked.

A function is invoked by referencing it wherever a value of the same type as the return type can be used.

Errors: X must be a positive, nonzero value, or the function returns the value -1.0 and places the following message in the EASEL OS/2 EE errorlog:

LN: argument <= 0; returning -1.

See Also: Math Library
include Reference (Chapter 1)

LocalDate() Function

Action: Returns a string representation of the current date using the preferred local date format. The exact result of this function will depend upon the country in which the program is running and on user preferences.

LocalDate(TYPE)

Where: TYPE must be either **PMDate** or **OS2Date**. If **PMDate** is specified, the string returned is formatted according to the format specified in the PM Control Panel "Country" settings. (Note that it is possible for the user to change this setting during the execution of the program.) If **OS2Date** is specified, the string returned is formatted according to the COUNTRY setting in the OS/2 CONFIG.SYS file.

Return Type: **string**

Include File: **datelib.inc**

Remarks: The include file, which contains the function declaration, must be referenced using the **include** statement before the function is invoked. The include file also contains the integer constant names **PMDate** and **OS2Date**.

A function is invoked by referencing it wherever a value of the same type as the return type can be used.

If an error is encountered, the returned string will be "ERROR".

Example: copy (LocalDate(PMDate)) to Date_String

LocalTime() Function

Action: Returns a string representation of the current time using the preferred local time format. The exact result of this function will depend upon the country in which the program is running and on user preferences.

LocalTime(TYPE)

Where: TYPE must be either **PMTime** or **OS2Time**. If **PMTime** is specified, the string returned is formatted according to the format specified in the PM Control Panel "Country" settings. (Note that it is possible for the user to change this setting during the execution of a program.) If **OS2Time** is specified, the string returned is formatted according to the COUNTRY setting in the OS/2 CONFIG.SYS file.

Return Type: **string**

Include File: **datelib.inc**

Remarks: The include file, which contains the function declaration, must be referenced using the **include** statement before the function is invoked. The include file also contains the integer constant names **PMTime** and **OS2Time**.

A function is invoked by referencing it wherever a value of the same type as the return type can be used.

If an error is encountered, the return string will be "ERROR".

The OS/2 12-hour time format does not include an AM/PM indicator. If you want AM/PM indicator in a 12-hour time format, use the **PMTime** format. The AM/PM indicator will be the string specified in the "Country" setting in the PM Control Panel.

Example: copy (LocalDate(PMTime)) to Time_String

LOG()

LOG() Function

Action: Returns the value of the logarithm base 10.

Model

LOG(X)

Where: X is a float value.

Return Type: **float**

Include File: **mathlib.inc**

Remarks: The include file, which contains the function declaration, must be referenced using the **include** statement before the function is invoked.

A function is invoked by referencing it wherever a value of the same type as the return type can be used.

Errors: X must be a positive, nonzero value, or the function returns the value -1.0 and places the following message in the EASEL OS/2 EE errorlog:

LOG: argument <= 0; returning -1.

See Also: Math Library
include Reference (Chapter 1)

MAX() Subroutine

Action: Returns the largest value in the float array. The value is returned in the MAXIMUM argument.

Model

MAX(ARRAY_IN, MAXIMUM)

Where: ARRAY_IN and MAXIMUM are float variables.

Include File: **mathlib.inc**

Remarks: The include file, which contains the subroutine declaration, must be referenced using the **include** statement before the subroutine is invoked.

A subroutine is invoked by referencing it in a **call** action statement.

You can determine the success or failure of the subroutine by testing the value of the **errorlevel** built-in function. A zero value indicates success; any nonzero value indicates failure.

See Also: Math Library
call Action Statement (Chapter 1)
errorlevel Action Inquiry Built-in Function (Chapter 1)
include Reference (Chapter 1)

MAXINT()

MAXINT() Subroutine

Action: Returns the largest value in the integer array. The value is returned in the MAXIMUM argument.

Model

MAXINT(ARRAY_IN, MAXIMUM)

Where: ARRAY_IN and MAXIMUM are integer variables.

Include File: **mathlib.inc**

Remarks: The include file, which contains the subroutine declaration, must be referenced using the **include** statement before the subroutine is invoked.

A subroutine is invoked by referencing it in a **call** action statement.

You can determine the success or failure of the subroutine by testing the value of the **errorlevel** built-in function. A zero value indicates success; any nonzero value indicates failure.

See Also: Math Library
call Action Statement (Chapter 1)
errorlevel Action Inquiry Built-in Function (Chapter 1)
include Reference (Chapter 1)

MEAN() Subroutine

Action: Returns the mean (or average) of all the values in the float array. The value is returned in the MEAN_VALUE argument.

Model

MEAN(ARRAY_IN, MEAN_VALUE)

Where: ARRAY_IN and MEAN_VALUE are float variables.

Include File: **mathlib.inc**

Remarks: The include file, which contains the subroutine declaration, must be referenced using the **include** statement before the subroutine is invoked.

A subroutine is invoked by referencing it in a **call** action statement.

You can determine the success or failure of the subroutine by testing the value of the **errorlevel** built-in function. A zero value indicates success; any nonzero value indicates failure.

See Also: Math Library
call Action Statement (Chapter 1)
errorlevel Action Inquiry Built-in Function (Chapter 1)
include Reference (Chapter 1)

MEANINT()

MEANINT() Subroutine

Action: Returns the mean (or average) of all the values in the integer array. The value is returned in the MEAN_VALUE argument.

Model

MEANINT(ARRAY_IN, MEAN_VALUE)

Where: ARRAY_IN and MEAN_VALUE are float variables.

Include File: **mathlib.inc**

Remarks: The include file, which contains the subroutine declaration, must be referenced using the **include** statement before the subroutine is invoked.

A subroutine is invoked by referencing it in a **call** action statement.

You can determine the success or failure of the subroutine by testing the value of the **errorlevel** built-in function. A zero value indicates success; any nonzero value indicates failure.

See Also: Math Library
call Action Statement (Chapter 1)
errorlevel Action Inquiry Built-in Function (Chapter 1)
include Reference (Chapter 1)

MEDIAN() Subroutine

Action: Returns the median of all the values in the float array. The value is returned in the **MEDIAN_VALUE** argument.

Model

MEDIAN(ARRAY_IN, MEDIAN_VALUE)

Where: **ARRAY_IN** and **MEDIAN_VALUE** are float variables.

Include File: **mathlib.inc**

Remarks: The include file, which contains the subroutine declaration, must be referenced using the **include** statement before the subroutine is invoked.

A subroutine is invoked by referencing it in a **call** action statement.

You can determine the success or failure of the subroutine by testing the value of the **errorlevel** built-in function. A zero value indicates success; any nonzero value indicates failure.

The median is defined as follows:

For an odd number of elements in an array, it is the middle value when all elements are sorted by value.

For an even number of elements in the array, the median is the average of the two middle values after all elements in the array are sorted by value.

Errors: If there is insufficient memory to allocate an array buffer, the subroutine sets **MEDIAN_VALUE** to -1 and places the following message in the EASEL OS/2 EE errorlog:

MEDIAN: memory allocation error; setting value to -1.

See Also: Math Library
call Action Statement (Chapter 1)
errorlevel Action Inquiry Built-in Function (Chapter 1)
include Reference (Chapter 1)

MEDIANINT() Subroutine

Action: Returns the median of all the values in the integer array. The value is returned in the MEDIAN_VALUE argument.

Model

MEDIANINT(ARRAY_IN, MEDIAN_VALUE)

Where: ARRAY_IN and MEDIAN_VALUE are integer variables.

Include File: **mathlib.inc**

Remarks: The include file, which contains the subroutine declaration, must be referenced using the **include** statement before the subroutine is invoked.

A subroutine is invoked by referencing it in a **call** action statement.

You can determine the success or failure of the subroutine by testing the value of the **errorlevel** built-in function. A zero value indicates success; any nonzero value indicates failure.

The median is defined as follows:

For an odd number of elements in an array, it is the middle value when all elements are sorted by value.

For an even number of elements in the array, the median is the average of the two middle values after all elements in the array are sorted by value.

Errors: If there is insufficient memory to allocate an array buffer, the subroutine sets MEDIAN_VALUE to -1 and places the following message in the EASEL OS/2 EE errorlog:

MEDIANINT: memory allocation error; setting value to -1.

See Also: Math Library
call Action Statement (Chapter 1)
errorlevel Action Inquiry Built-in Function (Chapter 1)
include Reference (Chapter 1)

MIN() Subroutine

Action: Returns the smallest value in the float array. The value is returned in the MINIMUM argument.

Model

MIN(ARRAY_IN, MINIMUM **)**

Where: ARRAY_IN and MINIMUM are integer variables.

Include File: **mathlib.inc**

Remarks: The include file, which contains the subroutine declaration, must be referenced using the **include** statement before the subroutine is invoked.

A subroutine is invoked by referencing it in a **call** action statement.

You can determine the success or failure of the subroutine by testing the value of the **errorlevel** built-in function. A zero value indicates success; any nonzero value indicates failure.

See Also: Math Library
call Action Statement (Chapter 1)
errorlevel Action Inquiry Built-in Function (Chapter 1)
include Reference (Chapter 1)

MININT()

MININT() Subroutine

Action: Returns the smallest value in the integer array. The value is returned in the MINIMUM argument.

Model

MININT(ARRAY_IN, MINIMUM)

Where: ARRAY_IN and MINIMUM are integer variables.

Include File: **mathlib.inc**

Remarks: The include file, which contains the subroutine declaration, must be referenced using the **include** statement before the subroutine is invoked.

A subroutine is invoked by referencing it in a **call** action statement.

You can determine the success or failure of the subroutine by testing the value of the **errorlevel** built-in function. A zero value indicates success; any nonzero value indicates failure.

See Also: Math Library
call Action Statement (Chapter 1)
errorlevel Action Inquiry Built-in Function (Chapter 1)
include Reference (Chapter 1)

MOD() Function

Action: Returns the remainder produced after dividing X by Y.

Model

MOD(X, Y)

Where: X and Y are float values.

Return Type: **float**

Include File: **mathlib.inc**

Remarks: The include file, which contains the function declaration, must be referenced using the **include** statement before the function is invoked.

A function is invoked by referencing it wherever a value of the same type as the return type can be used.

Examples of the MOD function: MOD(10.0, 3.0) returns 1.0, and MOD(22.0, 2.0) returns 0.0.

Errors: Y cannot equal 0.0, or the function returns the value -1 and places the following message in the EASEL OS/2 EE errorlog:

MOD: second argument = 0; returning -1.

See Also: Math Library
include Reference (Chapter 1)

OpenFile() Subroutine

Action: Performs the necessary overhead required when opening a file.

Model

OpenFile(FILE_ID, FILE_NAME, ACCESS_MODE)

Where: FILE_ID is a user-defined name that follows the rules for an EASEL OS/2 EE identifier. It must be declared previously by the **fileid** statement.

FILE_NAME is a string variable that is the name of the file to be opened.

ACCESS_MODE is a string variable that contains one of three file modes, indicating how the file can be accessed after it is opened: "**read-only**", "**write-only**", or "**append**".

Include File: **fileio.inc**

Remarks: The include file, which contains the subroutine declaration, must be referenced using the **include** statement before the subroutine is invoked.

A subroutine is invoked by referencing it in a **call** action statement.

You can determine the success or failure of the subroutine by testing the value of the **errorlevel** built-in function. A zero value indicates success; any nonzero value indicates failure.

If a file is opened twice for output, the one opened last will be the one that is actually written. Opening a file twice for reading is acceptable, as long as each call to **OpenFile()** specifies a unique file identifier. Opening a file for both reading and writing will produce different results depending on which one is opened and closed first.

<i>Errors:</i>	errorlevel	Error Message/Explanation
	1	Error locking global memory. Indicates a serious problem. Contact your support representative.
	3	Unrecognized file mode specified. Make sure file access mode begins with either "r", "w", or "a".
	4	Attempt to open a file that does not exist.
	5	Attempt to open write-protected file with mode "w". Check file's access permission.
	6	Attempt to open write-protected file with mode "a". Check file's access permission.
	7	Insufficient memory to satisfy allocation request. System memory is too low to allocate.
	8	File handle is invalid. Contact your support representative.
	24	File has already been opened. This error occurs when you call OpenFile() twice using the same file identifier. Either use a unique file identifier for each call to OpenFile() , or call CloseFile() for that file identifier before you open it again.

See Also: **File I/O Library**
call Action Statement (Chapter 1)
errorlevel Action Inquiry Built-in Function (Chapter 1)
fileid Definition (Chapter 1)
include Reference (Chapter 1)

PI()

PI() Function

Action: Returns the constant value of Pi, which is 3.14159...

Model

PI()

Return Type: **float**

Include File: **mathlib.inc**

Remarks: The include file, which contains the function declaration, must be referenced using the **include** statement before the function is invoked.

A function is invoked by referencing it wherever a value of the same type as the return type can be used.

See Also: Math Library
include Reference (Chapter 1)

POWER() Function

Action: Returns X raised to the Y power.

Model

POWER(X, Y)

Where: X and Y are float values.

Return Type: **float**

Include File: **mathlib.inc**

Remarks: The include file, which contains the function declaration, must be referenced using the **include** statement before the function is invoked.

A function is invoked by referencing it wherever a value of the same type as the return type can be used.

If X is negative and Y is not a whole number (i.e., Y has a nonzero fractional part), the function computes the value using only the integral part of Y.

If Y is 0.0, the function returns 1.0.

Errors: X cannot be 0.0 and Y cannot be negative, or the function returns the value 0.0 and places the following message in the EASEL OS/2 EE errorlog:

POWER: base = 0 and exponent < 0; returning 0.

See Also: Math Library
include Reference (Chapter 1)

ReadLineNumber() Subroutine

Action: Reads the specified line of an ASCII file, and stores the line in a single variable.

Model

ReadLineNumber(FILE_ID, LINE_NUMBER, TARGET)

Where: FILE_ID is a user-defined name that follows the rules for an EASEL OS/2 EE identifier. It must be declared previously by the **fileid** statement.

LINE_NUMBER is an integer variable that contains the number of the line to be read. This parameter must be an integer variable greater than zero.

TARGET is a string variable that receives the data from the appropriate input line of the file. The target string will not contain any linefeeds (\n) or carriage returns (\r).

Include File: **fileio.inc**

Remarks: The include file, which contains the subroutine declaration, must be referenced using the **include** statement before the subroutine is invoked.

A subroutine is invoked by referencing it in a **call** action statement.

You can determine the success or failure of the subroutine by testing the value of the **errorlevel** built-in function. A zero value indicates success; any nonzero value indicates failure.

ReadLineNumber() finds the requested line, if it is in the file, according to its position from the beginning of the file.

See Also: File I/O Library
call Action Statement (Chapter 1)
errorlevel Action Inquiry Built-in Function (Chapter 1)
fileid Definition (Chapter 1)
include Reference (Chapter 1)

ReadNext() Subroutine

Action: Reads the next line of an ASCII file sequentially, and stores the line in a single variable.

Model

ReadNext(FILE_ID, TARGET)

Where: FILE_ID is a user-defined name that follows the rules for an EASEL OS/2 EE identifier. It must be declared previously by the **fileid** statement.

TARGET is a string variable that receives the data from the appropriate input line of the file. The target string will not contain any linefeeds (\n) or carriage returns (\r).

Include File: **fileio.inc**

Remarks: The include file, which contains the subroutine declaration, must be referenced using the **include** statement before the subroutine is invoked.

A subroutine is invoked by referencing it in a **call** action statement.

You can determine the success or failure of the subroutine by testing the value of the **errorlevel** built-in function. A zero value indicates success; any nonzero value indicates failure.

ReadNext() can be called at any time, provided there are still lines in the file to be read.

See Also: File I/O Library
call Action Statement (Chapter 1)
errorlevel Action Inquiry Built-in Function (Chapter 1)
include Reference (Chapter 1)

ReadNextRecord()

ReadNextRecord() Subroutine

Action: Reads the next line of an ASCII file sequentially, and stores the line in a record.

Model

ReadNextRecord(FILE_ID, RECORD_NAME)

Where: FILE_ID is a user-defined name that follows the rules for an EASEL OS/2 EE identifier. It must be declared previously by the **fileid** statement.

RECORD_NAME is the name of a record. RECORD_NAME must be previously defined by the **record** definition statement.

Include File: **fileio.inc**

Remarks: The include file, which contains the subroutine declaration, must be referenced using the **include** statement before the subroutine is invoked.

A subroutine is invoked by referencing it in a **call** action statement.

You can determine the success or failure of the subroutine by testing the value of the **errorlevel** built-in function. A zero value indicates success; any nonzero value indicates failure.

ReadNextRecord() can be called at any time, provided there are still lines in the file to be read.

See Also: File I/O Library
call Action Statement (Chapter 1)
errorlevel Action Inquiry Built-in Function (Chapter 1)
fileid Definition (Chapter 1)
include Reference (Chapter 1)
record Definition (Chapter 1)

ReadRecordAtLine() Subroutine

Action: Reads the specified line of an ASCII file, and stores the line in a record.

Model

ReadRecordAtLine(FILE_ID, LINE_NUMBER, RECORD_NAME)

Where: FILE_ID is a user-defined name that follows the rules for an EASEL OS/2 EE identifier. It must be declared previously by the **fileid** statement.

LINE_NUMBER is an integer variable that contains the number of the line to be read. This parameter must be an integer variable greater than zero.

RECORD_NAME is the name of a record. RECORD_NAME must be previously defined by the **record** definition statement.

Include File: **fileio.inc**

Remarks: The include file, which contains the subroutine declaration, must be referenced using the **include** statement before the subroutine is invoked.

A subroutine is invoked by referencing it in a **call** action statement.

You can determine the success or failure of the subroutine by testing the value of the **errorlevel** built-in function. A zero value indicates success; any nonzero value indicates failure.

ReadRecordAtLine() finds the requested line, if it is in the file, according to its position from the beginning of the file.

See Also: File I/O Library
call Action Statement (Chapter 1)
errorlevel Action Inquiry Built-in Function (Chapter 1)
fileid Definition (Chapter 1)
include Reference (Chapter 1)
record Definition (Chapter 1)

ReplyToMessage()

ReplyToMessage() Function

Action: Displays a message box, then returns the option selected by the user.

Model

ReplyToMessage(TITLE, MESSAGE, OPTIONS, DEFAULT_OPTION, ICON)

Where: TITLE is a string value representing the title that will appear in the message box's title bar.

MESSAGE is a string value representing the message that will appear in the client area of the message box. Presentation Manager will word wrap the message and determine the size of the message box automatically.

OPTIONS is an integer value representing the push button options that will be provided in the message box.

DEFAULT_OPTION is an integer value representing which push button option is the default push button.

ICON is an integer value representing the icon to be displayed in the message box to indicate the type of message.

Return Type: **string**

Include File: **message.inc**

Remarks: The include file, which contains the function declaration, must be referenced using the **include** statement before the function is invoked. The include file also contains integer constant names for use with the OPTION and ICON arguments.

A function is invoked by referencing it wherever a value of the same type as the return type can be used.

Valid Options: **MessageOk**
MessageOkCancel
MessageRetryCancel
MessageAbortRetryIgnore
MessageYesNo
MessageYesNoCancel

Valid Icons: **MessageQuery**
 MessageWarning
 MessageInformation
 MessageCritical
 MessageNoIcon

Return **"yes"**
Values: **"no"**
 "abort"
 "retry"
 "cancel"
 "ignore"
 "ok"
 "error"

See Also: **Message Library**
 include Reference (Chapter 1)

SetBufferSize()

SetBufferSize() Subroutine

Action: Modifies the buffer size used by the File I/O Input subroutines.

Model

SetBufferSize(FILE_ID, BUFFER_SIZE)

Where: FILE_ID is a user-defined name that follows the rules for an EASEL OS/2 EE identifier. It must be declared previously by the **fileid** statement.

BUFFER_SIZE is an integer variable containing the buffer size in bytes.

Include File: **fileio.inc**

Remarks: The include file, which contains the subroutine declaration, must be referenced using the **include** statement before the subroutine is invoked. A subroutine is invoked by referencing it in a **call** action statement.

You can determine the success or failure of the subroutine by testing the value of the **errorlevel** built-in function. A zero value indicates success; any nonzero value indicates failure.

Increasing the buffer size will improve speed, but always increase the buffer size to a multiple of 1K, or response time may actually be slowed. The buffer size can be a maximum of 64K. If a request is made for more than 64K, a buffer of 64K will be allocated.

Default: 1K (1024 bytes).

See Also: File I/O Library
call Action Statement (Chapter 1)
errorlevel Action Inquiry Built-in Function (Chapter 1)
fileid Definition (Chapter 1)
include Reference (Chapter 1)

SetIndexSize() Subroutine

Action: Modifies the index size used by the File I/O Input subroutines to keep track of the number of lines in a file.

Model

SetIndexSize(FILE_ID, INDEX_SIZE)

Where: FILE_ID is a user-defined name that follows the rules for an EASEL OS/2 EE identifier. It must be declared previously by the **fileid** statement.

INDEX_SIZE is an integer variable containing the index size in bytes.

Include File: **fileio.inc**

Remarks: The include file, which contains the subroutine declaration, must be referenced using the **include** statement before the subroutine is invoked.

A subroutine is invoked by referencing it in a **call** action statement.

You can determine the success or failure of the subroutine by testing the value of the **errorlevel** built-in function. A zero value indicates success; any nonzero value indicates failure.

Each index requires 4 bytes.

Changing the index size does not affect speed when reading a file sequentially. It does affect speed when reading a file randomly.

Default: 1K (1024 bytes), or 256 entries.

See Also: File I/O Library
call Action Statement (Chapter 1)
errorlevel Action Inquiry Built-in Function (Chapter 1)
fileid Definition (Chapter 1)
include Reference (Chapter 1)

SetTabSize()

SetTabSize() Subroutine

Action: Controls the number of columns between tab stops.

Model

SetTabSize(FILE_ID, TAB_SIZE)

Where: FILE_ID is a user-defined name that follows the rules for an EASEL OS/2 EE identifier. It must be declared previously by the **fileid** statement.

TAB_SIZE is an integer variable containing the number of columns between tab stops (greater than 0 and less than 64K).

Include File: **fileio.inc**

Remarks: The include file, which contains the subroutine declaration, must be referenced using the **include** statement before the subroutine is invoked.

A subroutine is invoked by referencing it in a **call** action statement.

You can determine the success or failure of the subroutine by testing the value of the **errorlevel** built-in function. A zero value indicates success; any nonzero value indicates failure.

Default: 8 columns.

See Also: File I/O Library
call Action Statement (Chapter 1)
errorlevel Action Inquiry Built-in Function (Chapter 1)
fileid Definition (Chapter 1)
include Reference (Chapter 1)

SIN() Function

Action: Returns the sine of the argument in degrees.

Model

SIN(X)

Where: X is a float value.

Return Type: **float**

Include File: **mathlib.inc**

Remarks: The include file, which contains the function declaration, must be referenced using the **include** statement before the function is invoked.

A function is invoked by referencing it wherever a value of the same type as the return type can be used.

To convert degrees to radians, multiply degrees by 0.0174532926.

See Also: Math Library
include Reference (Chapter 1)

STDDEV() Subroutine

Action: Returns the standard deviation of all values in the float array. The value is returned in the STANDARD_DEV argument.

Model

STDDEV(ARRAY_IN, STANDARD_DEV)

Where: ARRAY_IN and STANDARD_DEV are float variables.

Include File: **mathlib.inc**

Remarks: The include file, which contains the subroutine declaration, must be referenced using the **include** statement before the subroutine is invoked.

A subroutine is invoked by referencing it in a **call** action statement.

You can determine the success or failure of the subroutine by testing the value of the **errorlevel** built-in function. A zero value indicates success; any nonzero value indicates failure.

See Also: Math Library
call Action Statement (Chapter 1)
errorlevel Action Inquiry Built-in Function (Chapter 1)
include Reference (Chapter 1)

STDDEVINT() Subroutine

Action: Returns the standard deviation of all values in the integer array. The value is returned in the STANDARD_DEV argument.

Model

STDDEVINT(ARRAY_IN, STANDARD_DEV)

Where: ARRAY_IN and STANDARD_DEV are float variables.

Include File: **mathlib.inc**

Remarks: The include file, which contains the subroutine declaration, must be referenced using the **include** statement before the subroutine is invoked.

A subroutine is invoked by referencing it in a **call** action statement.

You can determine the success or failure of the subroutine by testing the value of the **errorlevel** built-in function. A zero value indicates success; any nonzero value indicates failure.

See Also: Math Library
call Action Statement (Chapter 1)
errorlevel Action Inquiry Built-in Function (Chapter 1)
include Reference (Chapter 1)

SQRT()

SQRT() Function

Action: Returns the square root of X.

Model

SQRT(X)

Where: X is a float value.

Return Type: **float**

Include File: **mathlib.inc**

Remarks: The include file, which contains the function declaration, must be referenced using the **include** statement before the function is invoked.

A function is invoked by referencing it wherever a value of the same type as the return type can be used.

Errors: X cannot be negative, or the function returns the value 0 and places the following message in the EASEL OS/2 EE errorlog:

SQRT: argument < 0; returning 0.

See Also: Math Library
include Reference (Chapter 1)

SUM() Subroutine

Action: Returns the sum of all values in the float array. The value is returned in the SUMMATION argument.

Model

SUM(ARRAY_IN, SUMMATION)

Where: ARRAY_IN and SUMMATION are float variables.

Include File: **mathlib.inc**

Remarks: The include file, which contains the subroutine declaration, must be referenced using the **include** statement before the subroutine is invoked.

A subroutine is invoked by referencing it in a **call** action statement.

You can determine the success or failure of the subroutine by testing the value of the **errorlevel** built-in function. A zero value indicates success; any nonzero value indicates failure.

See Also: Math Library
call Action Statement (Chapter 1)
errorlevel Action Inquiry Built-in Function (Chapter 1)
include Reference (Chapter 1)

SUMINT()

SUMINT() Subroutine

Action: Returns the sum of all values in the integer array. The value is returned in the SUMMATION argument.

Model

SUMINT(ARRAY_IN, SUMMATION)

Where: ARRAY_IN and SUMMATION are integer variables.

Include File: **mathlib.inc**

Remarks: The include file, which contains the subroutine declaration, must be referenced using the **include** statement before the subroutine is invoked.

A subroutine is invoked by referencing it in a **call** action statement.

You can determine the success or failure of the subroutine by testing the value of the **errorlevel** built-in function. A zero value indicates success; any nonzero value indicates failure.

Errors: If the result exceeds the largest integer (2147483647) or the smallest integer (-2147483648), the subroutine sets SUMMATION to 0 and places one of the following messages in the EASEL OS/2 EE errorlog:

result > largest integer; setting value to 0.

result < smallest integer; setting value to 0.

See Also: Math Library
call Action Statement (Chapter 1)
errorlevel Action Inquiry Built-in Function (Chapter 1)
include Reference (Chapter 1)

TAN() Function

Action: Returns the tangent of the argument in degrees.

Model

TAN(X)

Return Type: X is a float value.

Include File: **mathlib.inc**

Remarks: The include file, which contains the function declaration, must be referenced using the **include** statement before the function is invoked.

A function is invoked by referencing it wherever a value of the same type as the return type can be used.

To convert degrees to radians, multiply degrees by 0.0174532926.

Errors: The argument must not be 90 or 270 degrees, or the function returns the value 0 and places the following message in the EASEL OS/2 EE errorlog:

TAN: not defined for 90 or 270 degrees; returning 0.

See Also: Math Library
include Reference (Chapter 1)

Validdate()

Validdate() Function

Action: Returns a boolean value that tells whether a given date is valid.

Model

Validdate(YEAR, MONTH, DAY)

Where: YEAR is an integer value representing the full year of the given date, i.e., 1989, 1990, etc.

MONTH is an integer value representing the month of the given date, i.e., 2 for February, 3 for March, etc.

DAY is an integer value representing the day of the given date.

Return Type: **boolean**

Include File: **datelib.inc**

Remarks: The include file, which contains the function declaration, must be referenced using the **include** statement before the function is invoked.

A function is invoked by referencing it wherever a value of the same type as the return type can be used.

Example: if (Validdate(1992, 2, 29)) then
 action GetData
 end if

See Also: Date Library
 include Reference (Chapter 1)

VARIANCE() Subroutine

Action: Returns the variance of all values in the float array. The value is returned in the VARIANCE argument.

Model

VARIANCE(ARRAY_IN, VARIANCE **)**

Where: ARRAY_IN and VARIANCE are float variables.

Include File: **mathlib.inc**

Remarks: The include file, which contains the subroutine declaration, must be referenced using the **include** statement before the subroutine is invoked.

A subroutine is invoked by referencing it in a **call** action statement.

You can determine the success or failure of the subroutine by testing the value of the **errorlevel** built-in function. A zero value indicates success; any nonzero value indicates failure.

The equation used is based upon the number of elements in the array, n, and not (n-1).

See Also: Math Library
call Action Statement (Chapter 1)
errorlevel Action Inquiry Built-in Function (Chapter 1)
include Reference (Chapter 1)

VARIANCEINT() Subroutine

Action: Returns the variance of all values in the integer array. The value is returned in the VARIANCE argument.

Model

VARIANCEINT(ARRAY_IN, VARIANCE)

Where: ARRAY_IN and VARIANCE are integer variables.

Include File: **mathlib.inc**

Remarks: The include file, which contains the subroutine declaration, must be referenced using the **include** statement before the subroutine is invoked.

A subroutine is invoked by referencing it in a **call** action statement.

You can determine the success or failure of the subroutine by testing the value of the **errorlevel** built-in function. A zero value indicates success; any nonzero value indicates failure.

The equation used is based upon the number of elements in the array, n, and not (n-1).

See Also: Math Library
call Action Statement (Chapter 1)
errorlevel Action Inquiry Built-in Function (Chapter 1)
include Reference (Chapter 1)

WeekDay() Function

Action: Returns an uppercase string that is the day of the week corresponding to the specified date.

Model

WeekDay(YEAR, MONTH, DAY)

Where: YEAR is an integer value representing the full year of the given date, i.e., 1989, 1990, etc.

MONTH is an integer value representing the month of the given date, i.e., 2 for February, 3 for March, etc.

DAY is an integer value representing the day of the given date.

Return Type: **string**

Include File: **datelib.inc**

Remarks: The include file, which contains the function declaration, must be referenced using the **include** statement before the function is invoked.

A function is invoked by referencing it wherever a value of the same type as the return type can be used. If an invalid date is specified, the NULL string is returned.

Example: copy (WeekDay(1989, 3, 8)) to Day_of_Week

See Also: Date Library
include Reference (Chapter 1)

WriteRecord()

WriteRecord() Subroutine

Action: Outputs the variables specified by the record definition to an ASCII file previously created by a call to **OpenFile()**.

Model

WriteRecord(FILE_ID, RECORD_NAME)

Where: FILE_ID is a user-defined name that follows the rules for an EASEL OS/2 EE identifier. It must be declared previously by the **fileid** statement.

RECORD_NAME is the name of the source record.

Include File: **fileio.inc**

Remarks: The include file, which contains the subroutine declaration, must be referenced using the **include** statement before the subroutine is invoked.

A subroutine is invoked by referencing it in a **call** action statement.

You can determine the success or failure of the subroutine by testing the value of the **errorlevel** built-in function. A zero value indicates success; any nonzero value indicates failure.

The source record can have strings that have embedded linefeeds as fields, so that a single call to the **WriteRecord()** subroutine can result in several lines being written to the ASCII file.

See Also: File I/O Library
call Action Statement (Chapter 1)
errorlevel Action Inquiry Built-in Function (Chapter 1)
fileid Definition (Chapter 1)
include Reference (Chapter 1)
record Definition (Chapter 1)

WriteString() Subroutine

Action: Outputs a string value to an ASCII file previously created by a call to **OpenFile()**.

Model

WriteString(FILE_ID, SOURCE)

Where: FILE_ID is a user-defined name that follows the rules for an EASEL OS/2 EE identifier. It must be declared previously by the **fileid** statement.

SOURCE is a string variable that can contain linefeeds (**\n**) and carriage returns (**\r**).

Include File: **fileio.inc**

Remarks: The include file, which contains the subroutine declaration, must be referenced using the **include** statement before the subroutine is invoked.

A subroutine is invoked by referencing it in a **call** action statement.

You can determine the success or failure of the subroutine by testing the value of the **errorlevel** built-in function. A zero value indicates success; any nonzero value indicates failure.

The source string can have embedded linefeeds, so that a single call to the **WriteString()** subroutine can result in several lines being written to the ASCII file.

See Also: File I/O Library
call Action Statement (Chapter 1)
errorlevel Action Inquiry Built-in Function (Chapter 1)
fileid Definition (Chapter 1)
include Reference (Chapter 1)

Chapter 3. Environment Files

Overview

This chapter describes the environment files used with IBM EASEL Version 1.1 for OS/2 Extended Edition (EASEL OS/2 EE) to compile and execute EASEL OS/2 EE graphical user interface applications.

COMPILE.CMD OS/2 Command File

Purpose: To specify the commands and arguments that are executed when compiling an EASEL OS/2 EE program.

Model

compile PROGRAM_NAME [COMMAND_LINE_OPTION]...

Where: PROGRAM_NAME is the name of the input file, containing EASEL OS/2 EE source code, to be compiled. This file must have an extension of .EAL.

COMMAND_LINE_OPTION is any combination of the command line options described below.

Command Line Options:

- b PROGRAM_NAME.ebi**
PROGRAM_NAME is the name EASEL OS/2 EE should use as the output file name. It will contain the binary version of the program after a successful compile. If this option is not specified, the .EAL name specified will be used and the .EAL extension will be replaced with .EBI.
Initial value: not specified
- debug**
If specified, must be in both COMPILE.CMD and EASEL.CMD files. Causes the line numbers from the .EAL files to be placed in the errorlog when traces and errors are generated during runtime.
Initial value: not specified
- e FILENAME**
Sends error messages to FILENAME.
Initial value: not specified

-f FILENAME

Defines which configuration file EASEL OS/2 EE is to use.

Initial value: not specified. By default, the filename is CONFIG.ESL, and it is searched for in the directories in the OS/2 PATH specification.

-fullscreen

Specifies that the application is fullscreen. It must be used if the application does not run in a window. Fullscreen applications do not allow the definition of a primary region. Running a fullscreen application causes the EGA/VGA color palette to be set to the fullscreen color set. Specifying the fullscreen flag also affects the way selectability works; in a fullscreen application, regions are disabled by default, and a region's selectability does not affect the selectability of its children.

-g SIZE

Specifies size, in bytes, of allocated global memory.

Initial value: not specified. This overrides the value of GLBSIZE specified in the configuration file CONFIG.ESL.

-ix

Places the EASEL OS/2 EE program text into the errorlog as the program is compiled. When using this option, the **-e** option should also be used. (Note that the "i" specified is the alphabetic character, not the numeric character "1".)

Initial value: not specified

-m SIZE

Specifies initial size, in bytes, of EASEL OS/2 EE program memory.

Initial value: not specified. This overrides the value of PRGSize specified in the configuration file CONFIG.ESL.

Remarks: When you compile an EASEL OS/2 EE program, EASEL OS/2 EE transforms the .EAL program statements into a binary representation contained in an OS/2 file, whose name will have the extension .EBI. The .EBI files are the executable interface. The compilation will also produce an errorlog file.

The command line options can be specified when executing the COMPILE.CMD file for a temporary change to the program execution. To have the COMPILE.CMD file default to using particular command line options, the EPARSE line in the COMPILE.CMD file can be modified to include any of the command line options. These additional default command line options should be inserted between EPARSE and %1. EPARSE is the program that the COMPILE.CMD file executes to transform EASEL OS/2 EE program statements into an EASEL OS/2 EE executable user interface. The command line options added to the EPARSE line in the COMPILE.CMD file can be temporarily changed by specifying the new value when executing the COMPILE.CMD file.

Example: compile myprog -m 150000 -e errors

See Also: **EASEL.CMD** OS/2 Command File

CONFIG.ESL File

Purpose: To specify the parameters that configure EASEL OS/2 EE for your computer.

Remarks: You can make a permanent change to any option by entering the options and values in the CONFIG.ESL file, or you can make a temporary change to any option by entering `set OPTION=VALUE` on the OS/2 command line, which overrides the value specified in the CONFIG.ESL file. The SET command affects EASEL OS/2 EE executions until the option is reset or until the system is rebooted.

You can also make a permanent change to any option by entering the appropriate SET command in the EASEL.CMD file or COMPILE.CMD file.

Options that have path values are set by the installation procedure and should not be changed.

Do not remove any lines from the CONFIG.ESL file.

*Configuration
Options:*

BEEPFREQ = NUMBER

Tone frequency, in Hertz, of the pointing device sound for correct touches. The higher the number, the higher the frequency.

Initial value: 1250

BOOPFREQ = NUMBER

Tone frequency, in Hertz, of the pointing device sound for incorrect touches. The higher the number, the higher the frequency.

Initial value: 250

CODEPAGE = NUMBER

Code page to use when compiling and running the EASEL OS/2 EE program. This is set by the installation program. Available values are: 850 (recommended), 437, 860, 863, 865.

Initial value: 850

ESLERR = FILENAME

Name of source file for error messages added to the errorlog.

Initial value: c:\cuart-ee\eslerr.txt

EXETYPE = EXTENSION

Default file extension for executables.

Initial value: .exe

FONTCNT = NUMBER

Number of fonts that can be loaded into memory at any one time by an EASEL OS/2 EE program.

Initial value: 20

FONTPATH = PATH

Directories to search for font files.

Initial value: c:\cuart-ee\dl

CAUTION: Do not place a copy of ESLFONTS.FON file in your current directory.

GLBMAX = NUMBER

Maximum size of EASEL OS/2 EE global memory, in kilobytes. This must be larger than or equal to GLBSIZE.

Initial value: 50

GLBSIZE = NUMBER

Size of EASEL OS/2 EE global memory, in kilobytes. This value may be overridden by the value specified after the command line option **-g** (if specified) in the .CMD files.

Initial value: 10

HLDELAY = NUMBER

Highlight delay parameter. A value between 0 and 32000 can be specified. The larger the value, the longer EASEL OS/2 EE will flash a highlight.

Initial value: 8000

INCPATH = PATH

Directories to search for "include" files.

Initial value: c:\cuart-ee

INCTYPE = EXTENSION

Default file extension used for "include" files.

Initial value: .inc

KBDTABWID = NUMBER

Amount of spaces to translate a tab character into when typed from the keyboard.

Initial value: 0. This leaves the tab character instead of translating it to spaces.

LOCALDLL = NAME

Name of DLL for supporting local applications. This value is set by the installation procedure and should not be changed.

Initial value: local

MODPATH = PATH

Directories to search for modules.

Initial value: c:\cuart-ee\modules

POINTER = POINTER

EASEL OS/2 EE's default pointing device. Value can be **mouse** or **touchscreen**.

Initial value: mouse

PRGMAX = NUMBER

Maximum size of EASEL OS/2 EE program memory, in kilobytes. Must be larger than or equal to PRGSIZE.

Initial value: 500

PRGSIZE = NUMBER

Initial size of EASEL OS/2 EE program memory, in kilobytes. This value may be overridden by the value specified after the command line option **-m**, if specified.

Initial value: 300

REMOTE = STRING

Remote application settings. This string is inserted before the settings specified in the **start remote** action statement.

Initial value: 1200 noper stopbits 1 tandem

REMOTEDLL = NAME

Name of DLL for supporting remote application. This value is set by the installation process and should not be changed.

Initial value: ESLACDI

SLEEPMAX = NUMBER

Maximum number of seconds allowed for a **wait** action statement.

Initial value: 60

TABWIDTH = NUMBER

Standard number of columns per tab, during application input. If this value is 0, tabs are not converted to spaces in application input.

Initial value: 8

TEXTPATH = PATH

Directories to search when reading files into a textual region.

Initial value: not specified. Only current directory is searched.

TEXTTYPE = EXTENSION

Default extension for files read into and written to from textual regions.

Initial value: not specified. Looks for a file with no extension.

XMAX and YMAX

Set in the CONFIG.ESL file during installation. Do not modify.

Example: *In CONFIG.ESL file:*

```
ESLERR=C:\cuart-ee\eslerr.txt
EXETYPE=.exe
FONTPATH=C:\cuart-ee\dll
HLDELAY=8000
MODPATH=C:\.cuart-ee\modules
BEEPFREQ=1250
BOOPFREQ=250
TEXTYPE=.txt
```

On OS/2 command line or in EASEL.CMD file:

```
set BEEPFREQ=1000
set BOOPFREQ=200
```

EASEL.CMD OS/2 Command File

Purpose: To specify the commands and arguments that are executed when executing an EASEL OS/2 EE program.

Model

easel PROGRAM_NAME [COMMAND_LINE_OPTION]...

Where: PROGRAM_NAME is the name of the input file, containing the EASEL OS/2 EE executable user interface, to be executed.

COMMAND_LINE_OPTION is any combination of the command line options described below.

*Command
Line*

Options:

-apio

Messages tracing communication to and from applications are placed in the errorlog file. When using this option, the **-e** option should also be used. Used for debugging.

Initial value: not specified

-d VAR_NAME INITIAL_VALUE

Initializes EASEL OS/2 EE scalar (non-array) integer, float, boolean, and string variables. The variable must be declared in the program being started. The initialization occurs once, when EASEL OS/2 EE is started, and it overrides program initialization values. This option does not affect subsequent program changes, except that global variables retain their values. To initialize a string variable with a value containing more than one word, enclose the entire initialization value in double quotes ("). EASEL OS/2 EE attempts to convert the initial value to match the type of the variable. It is an error if (1) EASEL OS/2 EE determines that the types do not match, (2) there is no initial value, (3) the variable is undefined, (4) the variable is not one of the types listed above, or (5) the variable is an array.

Initial value: not specified

-debug

If specified, must be in both COMPILE.CMD and EASELCMD files. Causes the line numbers from the .EAL files to be placed in the errorlog when traces and errors are generated during runtime. When using this option, the **-e** option should also be used.

Initial value: not specified

-e FILENAME

Sends error messages to FILENAME.

Initial value: not specified

If FILENAME is not specified, the errors are not output.

-f FILENAME

Defines which configuration file EASEL OS/2 EE is to use.

Initial value: not specified. By default, the filename is CONFIG.ESL, and it is searched for in the directories in the OS/2 PATH specification.

-g SIZE

Specifies size, in bytes, of allocated global memory.

Initial value: not specified. This overrides the value of GLBSIZE specified in the configuration file CONFIG.ESL.

-m SIZE

Specifies size, in bytes, of allocated EASEL OS/2 EE program memory.

Initial value: not specified. This overrides the value of PRGSIZE specified in the configuration file CONFIG.ESL.

-nb

When specified, turns off pointing device sounds ("beep" for correct touches and "boop" for mis-touches).

Initial value: not specified. You can change the frequency of these sounds in the CONFIG.ESL file.

-terp

Causes EASEL OS/2 EE to generate trace messages for the entire EASEL OS/2 EE program or until a **turn trace off** action statement is encountered. These trace messages are useful for following EASEL OS/2 EE program execution at a low level. When using this option, the **-e** option should also be used. (The **-terp** option works in exactly the same way as the **turn trace on** action statement, except that **-terp** traces the entire program from the beginning, only stopping if a **turn trace off** statement is encountered.)

Initial value: not specified

EASEL.CMD

Remarks: The command line options can be specified when executing the EASEL.CMD file for a temporary change to the program execution. To have the EASEL.CMD file default to using particular command line options, the ESLRUN line in the EASEL.CMD file can be modified to include any of the command line options. These additional default command line options should be inserted between ESLRUN and %1. ESLRUN is the program that the EASEL.CMD file executes to run an EASEL OS/2 EE executable user interface. The command line options added to the ESLRUN line in the EASEL.CMD file can be temporarily changed by specifying the new value when executing the EASEL.CMD file.

Example: `cuart myprog -m 150000 -e errors`

See Also: **COMPILE.CMD** OS/2 Command File

Chapter 4. Include Files

Overview

This chapter shows the contents of the include files provided with IBM EASEL Version 1.1 for OS/2 Extended Edition (EASEL OS/2 EE).

accel.inc File

```
#####
#
# Name: ACCEL.INC
#
# Constants for defining accelerator keys.
#
# (C) Copyright Interactive Images, Inc. 1989, 1990
# All Rights Reserved
#
#####

integer constant KEY_BREAK      is 261
integer constant KEY_BACKSPACE is 262
integer constant KEY_TAB       is 263
integer constant KEY_BACKTAB   is 264
integer constant KEY_PAUSE     is 270
integer constant KEY_CAPSLOCK  is 271
integer constant KEY_ESC       is 272
integer constant KEY_SPACE     is 273
integer constant KEY_PAGEUP    is 274
integer constant KEY_PAGEDOWN  is 275
integer constant KEY_END       is 276
integer constant KEY_HOME      is 277
integer constant KEY_LEFT      is 278
integer constant KEY_UP        is 279
integer constant KEY_RIGHT     is 280
integer constant KEY_DOWN      is 281
integer constant KEY_PRINTSCRN is 282
integer constant KEY_INSERT    is 283
integer constant KEY_DELETE    is 284
integer constant KEY_SCRLLCK   is 285
integer constant KEY_NUMLOCK   is 286
integer constant KEY_ENTER     is 287
integer constant KEY_SYSRQ     is 288
integer constant KEY_F1        is 289
integer constant KEY_F2        is 290
integer constant KEY_F3        is 291
integer constant KEY_F4        is 292
integer constant KEY_F5        is 293
integer constant KEY_F6        is 294
integer constant KEY_F7        is 295
integer constant KEY_F8        is 296
integer constant KEY_F9        is 297
integer constant KEY_F10       is 298
integer constant KEY_F11       is 299
integer constant KEY_F12       is 300
```

datelib.inc File

```
#####
#
# Name: DATELIB.INC
#
# Declarations for day/date library routines.
#
# (C) Copyright Interactive Images, Inc. 1989, 1990
# All Rights Reserved
#
#####

integer constant PMDate is      1
integer constant OS2Date is    2
integer constant PMTime is     1
integer constant OS2Time is    2

function Validdate(integer:Year, integer:Month, integer:Day)
    returns boolean library "datelib"
function WeekDay(integer:Year, integer:Month, integer:Day)
    returns string library "datelib"
function DaySpan(integer:Year, integer:Month, integer:Day, integer:Year,
    integer:Month, integer:Day)
    returns integer library "datelib"
function LocalDate(integer:Type)
    returns string library "datelib"
function LocalTime(integer:Type)
    returns string library "datelib"
```

egacolor.inc File

```
#####
#
# Name: EGACOLOR.INC
#
# Declarations for full-screen EGA/VGA color palette functions.
#
# (C) Copyright Interactive Images, Inc. 1989, 1990
# All Rights Reserved
#
#####

# this file is now obsolete, and is maintained for backward
# compatibility, the contents of this file are now in esllib.inc

include "esllib.inc"
```

eslappc.inc File

```
#####
#
# Name: ESLAPPC.INC
#
# Declarations for APPC subroutines and constants
#
# (C) Copyright International Business Machines Corporation 1990.
# All Rights Reserved
#
#####

# define EASEL OS/2 EE APPC Subroutines callable from EASEL OS/2 EE

subroutine APPCAcceptStart(integer:Conversation_ID,
                          string:Profile_Name,
                          integer:State_Switch)
    library "eslappc"

subroutine APPCConvertA2E(integer:Conversation_ID,
                          string:String,
                          integer:Index_Start,
                          integer:Index_End,
                          integer:ASCII_Table,
                          integer:EBCDIC_Table)
    library "eslappc"

subroutine APPCConvertE2A(integer:Conversation_ID,
                          string:String,
                          integer:Index_Start,
                          integer:Index_End,
                          integer:ASCII_Table,
                          integer:EBCDIC_Table)
    library "eslappc"

subroutine APPCEnd(integer:Conversation_ID,
                   integer:Sync_Level)
    library "eslappc"

subroutine APPCFlush(integer:Conversation_ID)
    library "eslappc"

subroutine APPCGetAcceptProfile(string:Profile_Name,
                                string:Invocation_CMD,
                                integer:Inactivity_Timeout,
                                integer:Buffer_Size,
                                integer:ASCII_Table,
                                integer:EBCDIC_Table)
    library "eslappc"

subroutine APPCGetAcceptProfileNames(string:Profile_Names)
    library "eslappc"
```



```

subroutine APPCGetInfo(integer:Conversation_ID,
                      string:APPC_TP_ID,
                      integer:APPC_Conversation_ID,
                      string:NET_Name,
                      string:Local_LU_Name,
                      string:Local_LU_Alias_Name,
                      string:Partner_LU_Alias_Name,
                      string:Partner_LU_Uninterpreted_Name,
                      string:FullyQual_Partner_LU_Name,
                      string:Partner_TP_Name,
                      string:Mode_Name,
                      integer:Sync_Level,
                      string:User_ID)
    library "eslappc"

subroutine APPCGetInitProfile(string:Profile_Name,
                              string:Local_LU_Alias_Name,
                              string:Partner_LU_Alias_Name,
                              string:Partner_TP_Name,
                              string:Mode_Name,
                              string:Security_Parameters,
                              integer:Inactivity_Timeout,
                              integer:Buffer_Size,
                              integer:ASCII_Table,
                              integer:EBCDIC_Table)
    library "eslappc"

subroutine APPCGetInitProfileNames(string:Profile_Names)
    library "eslappc"

subroutine APPCGetString(integer:Conversation_ID,
                          string:String_Var,
                          integer:Auto_Convert)
    library "eslappc"

subroutine APPCGetSystemError(integer:Conversation_ID,
                              integer:Work_Request_Error,
                              integer:Work_Request_Identifier,
                              integer:APPC_Verb,
                              integer:Primary_Error_Code,
                              integer:Secondary_Error_Code)
    library "eslappc"

subroutine APPCReadyToReceive(integer:Conversation_ID)
    library "eslappc"

subroutine APPCReceiveNotify(integer:Conversation_ID)
    library "eslappc"

subroutine APPCRequestConfirm(integer:Conversation_ID)
    library "eslappc"

subroutine APPCRequestToSend(integer:Conversation_ID)
    library "eslappc"

```

eslappc.inc

```
subroutine APPCSendConfirmed(integer:Conversation_ID)
    library "eslappc"

subroutine APPCSendError(integer:Conversation_ID,
                        integer>Error_Origin)
    library "eslappc"

subroutine APPCSendString(integer:Conversation_ID,
                        string:String_Var,
                        integer:Auto_Convert,
                        integer:Send_Ind,
                        integer:More_to_Send)
    library "eslappc"

subroutine APPCSetAcceptProfile(string:Profile_Name,
                                integer:Action,
                                string:Invocation_CMD,
                                integer:Inactivity_Timeout,
                                integer:Buffer_Size,
                                integer:ASCII_Table,
                                integer:EBCDIC_Table)
    library "eslappc"

subroutine APPCSetInitProfile(string:Profile_Name,
                                integer:Action,
                                string:Local_LU_Alias_Name,
                                string:Partner_LU_Alias_Name,
                                string:Partner_TP_Name,
                                string:Mode_Name,
                                string:Security_Parameters,
                                integer:Inactivity_Timeout,
                                integer:Buffer_Size,
                                integer:ASCII_Table,
                                integer:EBCDIC_Table)
    library "eslappc"

subroutine APPCStart(integer:Conversation_ID,
                    string:Profile_Name,
                    integer:Sync_Level,
                    integer:State_Switch)
    library "eslappc"

subroutine APPCTestReqToSend(integer:Conversation_ID)
    library "eslappc"

# define EASEL OS/2 EE APPC Messages sent to EASEL OS/2 EE

integer constant APPC_Confirmed      is 5
integer constant APPC_ConfirmRequest is 4
integer constant APPC_Data           is 3
integer constant APPC_Ended          is 2
integer constant APPC_EndReceive     is 12
integer constant APPC_InputReceived  is 11
integer constant APPC_SendError      is 9
integer constant APPC_ReceiveError   is 8
integer constant APPC_RequestToSend is 6
integer constant APPC_SendState      is 7
```

```

integer constant APPC_Started          is 1
integer constant APPC_SystemError      is 10

# define EASEL OS/2 EE APPC Constants used in Subroutine Calls

integer constant APPC_Default          is 0
integer constant APPC_None             is 1
integer constant APPC_Confirm          is 2
integer constant APPC_Automatic        is 3
integer constant APPC_Controlled       is 4
integer constant APPC_NoConvert        is 5
integer constant APPC_Convert          is 6
integer constant APPC_Flush            is 7
integer constant APPC_SyncLevel        is 8
integer constant APPC_MoreData         is 9
integer constant APPC_ReadyToReceive   is 10
integer constant APPC_Abend            is 11
integer constant APPC_BadGet           is 12
integer constant APPC_BadSend          is 13
integer constant APPC_CreateProfile     is 14
integer constant APPC_ReplaceProfile    is 15
integer constant APPC_DeleteProfile     is 16

# Return Codes returned by the APPC... Subroutines

integer constant APPC_NORMAL_COMPLETION is 0

# General Errors
integer constant APPC_ERR_INFO_NOT_AVAILABLE is 5
integer constant APPC_ERR_STRING_TRUNCATED is 6
integer constant APPC_ERR_ON_FILE_OPEN is 8
integer constant APPC_ERR_ON_FILE_READ is 9
integer constant APPC_ERR_ON_FILE_WRITE is 10
integer constant APPC_ERR_OUT_OF_DISK_SPACE is 12
integer constant APPC_ERR_INV_WINDOW_HDL is 14
integer constant APPC_ERR_BUFFER_TOO_SMALL is 15
integer constant APPC_ERR_OUT_OF_MEMORY is 16
integer constant APPC_ERR_CONVERSION_ERROR is 18

# Input Parameter Errors
integer constant APPC_ERR_INVALID_PARMS is 20
integer constant APPC_ERR_INV_CONV_ID is 21
integer constant APPC_ERR_INV_SYNC_LEVEL is 22
integer constant APPC_ERR_INV_STATE_SWITCH is 23
integer constant APPC_ERR_INV_AUTO_CONVERT is 24
integer constant APPC_ERR_INV_SEND_IND is 25
integer constant APPC_ERR_INV_MORE_TO_SEND is 26
integer constant APPC_ERR_INV_ERROR_ORIGIN is 27
integer constant APPC_ERR_INV_INDEX_START is 28
integer constant APPC_ERR_INV_INDEX_END is 29
integer constant APPC_ERR_INV_INTEGER_VAR is 30
integer constant APPC_ERR_INV_STRING_VAR is 31
integer constant APPC_ERR_INV_PROFILE_NAME is 32
integer constant APPC_ERR_INV_LOC_LU_ALIAS_NAME is 33
integer constant APPC_ERR_INV_PAR_LU_ALIAS_NAME is 34
integer constant APPC_ERR_INV_PAR_TP_NAME is 35
integer constant APPC_ERR_INV_MODE_NAME is 36
integer constant APPC_ERR_INV_SEC_PARMS is 37
integer constant APPC_ERR_INV_INACT_TIMEOUT is 38
integer constant APPC_ERR_INV_BUFFER_SIZE is 39
integer constant APPC_ERR_INV_ASCII_TABLE is 40
integer constant APPC_ERR_INV_EBCDIC_TABLE is 41
integer constant APPC_ERR_INV_PROFILE_ACTION is 42

```

```
integer constant APPC_ERR_INV_INVOC_COMMAND is 43
integer constant APPC_ERR_INV_BUFFER_POINTER is 44
integer constant APPC_ERR_INV_BUFFER_LENGTH is 45
integer constant APPC_ERR_LGTH_EXCDS_PRF_BUFSIZE is 46
```

Request Processing Errors

```
integer constant APPC_ERR_PROFILE_ALREADY_EXISTS is 50
integer constant APPC_ERR_NO_DATA_AVAILABLE is 51
integer constant APPC_ERR_SYSTEM_ERROR_PENDING is 52
integer constant APPC_ERR_INV_TO_ISSUE_CNFRM_REQ is 53
integer constant APPC_ERR_INV_TO_ISSUE_CONFIRMED is 54
integer constant APPC_ERR_CONFIRM_REQ_IGNORED is 55
integer constant APPC_ERR_INV_TO_ISSUE_RECV_NTFY is 56
integer constant APPC_ERR_NO_REMOTE_TP_PARAM is 57
integer constant APPC_ERR_BASIC_CONV_NOT_VALID is 58
integer constant APPC_ERR_NO_SYSTEM_ERROR_DATA is 59
integer constant APPC_ERR_CONV_ENDING_REQ_PURGED is 60
```

Errors when calling EASEL OS/2 EE Subroutines

```
integer constant APPC_ERR_UNEXPECTED_EASEL_ERROR is 70
```

Unexpected miscellaneous errors

```
integer constant APPC_ERR_UNEXPECTED_ERROR is 80
```

Unexpected errors calling Dos... Subroutines

```
integer constant APPC_ERR_UNEXPECTED_DOS_ERROR is 90
```

Errors when calling APPC

```
integer constant APPC_ERROR is 100
```

Values placed in WORK_REQUEST_IDENTIFIER on return from APPCGetSystemErr

```
integer constant APPC_AcceptStart is 35
integer constant APPC_ConvertString is 37
integer constant APPC_GetInfo is 38
integer constant APPC_ReceiveBuffer is 30
integer constant APPC_ReceiveNotify is 25
integer constant APPC_ReceiveString is 28
integer constant APPC_RequestConfirm is 22
integer constant APPC_RequestRdyToReceive is 31
integer constant APPC_RequestSend is 26
integer constant APPC_SendAbend is 36
integer constant APPC_SendConfirmed is 23
integer constant APPC_SendData is 20
integer constant APPC_SendErrorItem is 24
integer constant APPC_SendFlush is 21
integer constant APPC_SendTerminate is 32
integer constant APPC_Start is 34
integer constant APPC_TestReqSend is 27
```

Values placed in APPC_VERB on return from APPCGetSystemError.

```
integer constant AP_M_ALLOCATE is 1
integer constant AP_M_CONFIRM is 3
integer constant AP_M_CONFIRMED is 4
integer constant AP_M_DEALLOCATE is 5
integer constant AP_M_FLUSH is 6
integer constant AP_M_GET_ATTRIBUTES is 7
integer constant AP_M_PREPARE_TO_RECEIVE is 10
integer constant AP_M_RECEIVE_AND_WAIT is 11
integer constant AP_M_RECEIVE_IMMEDIATE is 12
integer constant AP_M_RECEIVE_AND_POST is 13
integer constant AP_M_REQUEST_TO_SEND is 14
integer constant AP_M_SEND_DATA is 15
```

integer constant	AP_M_SEND_ERROR	is	16
integer constant	AP_M_TEST_RTS	is	18
integer constant	AP_RECEIVE_ALLOCATE	is	22
integer constant	AP_TP_ENDED	is	19
integer constant	AP_TP_STARTED	is	20
integer constant	SV_GET_CP_CONVERT_TABLE	is	25

esaudio.inc File

```
#####
#
# Name: ESLAUDIO.INC
#
# Declarations for audio functions.
#
# (C) Copyright International Business Machines Corporation 1990.
# All Rights Reserved
#
#####

#
# Audio Subroutine definitions
#

subroutine AUDClose      ( ) library "esaudio"

subroutine AUDErase      ( string : AUDIO_FILENAME ) library "esaudio"

subroutine AUDOpen       ( string : AUDIO_PATHNAME ) library "esaudio"

subroutine AUDPause      ( ) library "esaudio"

subroutine AUDPlay       ( string : AUDIO_FILENAME,
                           integer : PLAY_START_POSITION,
                           integer : PLAY_STOP_POSITION,
                           integer : VOLUME,
                           integer : BALANCE,
                           integer : BUSY_ACTION ) library "esaudio"

subroutine AUDQueryBusy  ( integer : BUSY_STATUS ) library "esaudio"

subroutine AUDQueryElapsedTime ( integer : ELAPSED_TIME ) library "esaudio"

subroutine AUDQueryPlayingTime ( string : AUDIO_FILENAME,
                                   integer : PLAYING_TIME ) library "esaudio"

subroutine AUDRecord     ( string : AUDIO_FILENAME,
                           integer : RECORD_START_POSITION,
                           integer : RECORD_STOP_POSITION,
                           integer : SOUND_SOURCE,
                           integer : SOUND_QUALITY,
                           integer : BEEP,
                           integer : BUSY_ACTION ) library "esaudio"

subroutine AUDResume     ( ) library "esaudio"

subroutine AUDSetBalance ( integer : BALANCE ) library "esaudio"

subroutine AUDSetVolume  ( integer : VOLUME ) library "esaudio"

subroutine AUDStop       ( integer : PURGE_ACTION ) library "esaudio"

# defaults
integer constant AUD_DefaultVolume is 100
```

```

integer constant AUD_DefaultBalance is 50
integer constant AUD_DefaultQuality is 1

# BUSY_ACTION values
integer constant AUD_Interrupt is 1 # interrupt current audio
integer constant AUD_Return is 2 # do not interrupt audio
integer constant AUD_Queue is 4 # queue audio request

# SOUND_SOURCE values
integer constant AUD_Mike is 0 # standard mike input
integer constant AUD_LineLeft is 1 # Line left channel only
integer constant AUD_LineRight is 2 # Line right channel only
integer constant AUD_Line is 3 # Line both channels
integer constant AUD_MikeLow is 4 # Low Mike input

# SOUND_QUALITY values
integer constant AUD_MusicQuality is 1 # 11k/sec
integer constant AUD_VoiceQuality is 2 # 5.5k/sec
integer constant AUD_StereoQuality is 3 # 22k/sec

# BEEP values
integer constant AUD_NoBeep is 0
integer constant AUD_Beep is 1

# PURGE_ACTION values
integer constant AUD_NoPurge is 0 # stop and do next audio req
integer constant AUD_Purge is 1 # stop and clear audio queue

# AUDQueryBusy return values
integer constant AUD_NotBusy is 0
integer constant AUD_BusyPlaying is 4
integer constant AUD_BusyRecording is 6
integer constant AUD_BusyPausing is 8

# Audio Stimulus messages (passed in "eventnumber")
integer constant AUD_Start is 3
integer constant AUD_End is 4
integer constant AUD_Error is 7

# AUD_Start and AUD_End parameter filters (passed in "eventparam")
integer constant AUD_Play is 1
integer constant AUD_Record is 2

# AUD_Error parameter filters (passed in "eventparam")
integer constant AUD_OpenError is 8
integer constant AUD_DiskFull is 22

# return codes
integer constant AUD_NORMAL_COMPLETION is 0
integer constant AUD_ERR_BUSY is 4
integer constant AUD_ERR_NOT_BUSY is 4
integer constant AUD_ERR_ALREADY_DONE is 4
integer constant AUD_ERR_OPEN is 8
integer constant AUD_ERR_QUEUE_FULL is 16
integer constant AUD_ERR_INVALID_PARAMS is 20

```

esllib.inc File

```
#####  
#  
# Name: ESLLIB.INC  
#  
# Declarations for miscellaneous esllib functions (full-screen  
# EGA/VGA color palette functions, and EslQueryFocus).  
#  
# (C) Copyright Interactive Images, Inc. 1990  
# All Rights Reserved  
#  
#####  
  
function EslSetEgaColor(integer: ColorNumber, integer: ColorValue)  
    returns integer  
    library "esllib"  
function EslQueryEgaColor(integer: ColorNumber)  
    returns integer  
    library "esllib"  
function EslQueryFocus()  
    returns string  
    library "esllib"
```

fileio.inc File

```
#####
#                                                                 #
# Name: FILEIO.INC                                             #
#                                                                 #
# Declarations for file input/output routines.                 #
#                                                                 #
# (C) Copyright Interactive Images, Inc. 1989, 1990           #
# All Rights Reserved                                         #
#                                                                 #
#####

subroutine OpenFile( fileid:FileID, string:FileName, string:FileMode
                    library "fileio"
subroutine ReadLineNumber( fileid:FileID, integer:LineNumber, string:Target )
                    library "fileio"
subroutine ReadNext( fileid:FileID, string:Target )          library "fileio"
subroutine ReadRecordAtLine( fileid:FileID, integer:LineNumber, record:Example )
                    library "fileio"
subroutine ReadNextRecord( fileid:FileID, record:Example )   library "fileio"
subroutine WriteFile( fileid:FileID, string:Source )         library "fileio"
subroutine WriteRecord( fileid:FileID, record:Example )      library "fileio"
subroutine CloseFile( fileid:FileID )                        library "fileio"
subroutine SetBufferSize( fileid:FileID, integer:BufferSize )
                    library "fileio"
subroutine SetIndexSize( fileid:FileID, integer:IndexSize )  library "fileio"
subroutine SetTabSize( fileid:FileID, integer:TabSize )      library "fileio"
subroutine GetError( integer>ErrorLevel, string:Message )    library "fileio"

#####
#                                                                 #
# The following declarations are available for compatibility    #
# with the release 1.0 names of these subroutines. If you wish #
# to use the following names, just uncomment the declarations. #
# (The library still contains the entry points.)                #
#                                                                 #
#####

#subroutine Open( fileid:FileID, string:FileName, string:FileMode )
#                                                                 library "fileio"
#subroutine Close( fileid:FileID )                             library "fileio"
#subroutine Write( fileid:FileID, string:Source )              library "fileio"
```

mathlib.inc File

```
#####
# Name: MATHLIB.INC
#
# Declarations for math library functions.
#
# (C) Copyright Interactive Images, Inc. 1989
# All Rights Reserved
#
#####

function SIN(float:X)           returns float library "mathlib"
function COS(float:X)           returns float library "mathlib"
function TAN(float:X)           returns float library "mathlib"
function ARCCOS(float:X)        returns float library "mathlib"
function ARCSIN(float:X)        returns float library "mathlib"
function ARCTAN(float:X)        returns float library "mathlib"
function ARCTAN2(float:X,
                    float:Y)     returns float library "mathlib"
function LN(float:X)            returns float library "mathlib"
function LOG(float:X)           returns float library "mathlib"
function EXP(float:X)           returns float library "mathlib"
function ABS(float:X)           returns float library "mathlib"
function MOD(float:X,
                    float:Y)     returns float library "mathlib"
function POWER(float:Mantissa,
               float:Exponential) returns float library "mathlib"
function Sqrt(float:X)          returns float library "mathlib"
function PI()                   returns float library "mathlib"
function E()                    returns float library "mathlib"
function FLOOR(float:X)         returns integer library "mathlib"
function CEIL(float:X)          returns integer library "mathlib"

subroutine MAX(float:Array,
               float:Value)      library "mathlib"
subroutine MAXINT(integer:Array,
                  integer:Value) library "mathlib"
subroutine MIN(float:Array,
               float:Value)      library "mathlib"
subroutine MININT(integer:Array,
                  integer:Value) library "mathlib"
subroutine MEAN(float:Array,
                float:Value)     library "mathlib"
subroutine MEANINT(integer:Array,
                   float:Value)  library "mathlib"
subroutine MEDIAN(float:Array,
                  float:Value)    library "mathlib"
subroutine MEDIANINT(integer:Array,
                     float:Value) library "mathlib"
subroutine VARIANCE(float:Array,
                    float:Value)  library "mathlib"
subroutine VARIANCEINT(integer:Array,
                       float:Value) library "mathlib"
subroutine STDDEV(float:Array,
                  float:Value)    library "mathlib"
subroutine STDDEVINT(integer:Array,
                     float:Value) library "mathlib"
subroutine SUM(float:Array,
```

```
float:Value)
subroutine SUMINT(integer:Array,
integer:Value)
```

```
library "mathlib"
library "mathlib"
```

message.inc File

```
#####
#
# Name: MESSAGE.INC
#
# Definitions to be used in calling ReplyToMessage
#
# (C) Copyright Interactive Images, Inc. 1989, 1990
# All Rights Reserved.
#
#####

#
# ReplyToMessage( Title, MessageText, Options, DefaultOption, Icon)
# returns Reply:String
#
function ReplyToMessage(
    string:Title,
    string:Message,
    integer:Options,
    integer:DefaultOption,
    integer:Icon)    returns string library "message"

#
# Title is a string. (This is displayed in the title bar of the message box.)
#
# MessageText is a string.
#
# Options is one of:
#     integer constant MessageOK           is 0
#                     MessageOKCancel     is 1
#                     MessageRetryCancel  is 2
#                     MessageAbortRetryIgnore is 3
#                     MessageYesNo        is 4
#                     MessageYesNoCancel  is 5
#
# DefaultOption is 1, 2, or 3. (For example, if it's a YesNo message, choose
# option 1 to make Yes the default, option 2 to make No the default and
# option 3 is meaningless.)
#
# Icon is one of:
#     integer constant MessageIconQuestion is 16
#                     MessageIconExclamation is 32
#                     MessageIconAsterisk is 48
#                     MessageIconHand is 64
#                     MessageNoIcon is 0
#
#                     MessageQuery is 16
#                     MessageWarning is 32
#                     MessageInformation is 48
#                     MessageCritical is 64
#
# Reply will be one of:
#
# "yes"
# "no"
# "abort"
# "retry"
# "cancel"
```

```
# "ignore"  
# "ok"  
#  
# "error"
```

pmptr.inc File

```
#####  
#  
# Name: PMPTR.INC  
#  
# Presentation Manager system pointers.  
#  
# (C) Copyright Interactive Images, Inc. 1989  
# All Rights Reserved  
#  
#####
```

```
integer constant  
SPTR_ARROW is 1 # Arrow cursor  
SPTR_TEXT is 2 # Text entry I-beam  
SPTR_WAIT is 3 # Hourglass  
SPTR_MOVE is 5 # Four-way arrow  
SPTR_SIZENWSE is 6 # Size border arrow for UL and LR corners  
SPTR_SIZENSW is 7 # Size border arrow for LL and UR corners  
SPTR_SIZELW is 8 # Size border arrow for left and right sides  
SPTR_SIZES is 9 # Size border arrow for top and bottom  
SPTR_APPICON is 10 # Large rectangle  
SPTR_ICONINFORMATION is 11 # Hand inside of a stop sign  
SPTR_ICONQUESTION is 12 # Question mark inside of inverted triangle  
SPTR_ICONERROR is 13 # Exclamation mark inside of a triangle  
SPTR_ICONWARNING is 14 # Large asterisk inside of a box  
SPTR_ILLEGAL is 18 # Diagonal line inside of a circle  
SPTR_FILE is 19 # Rectangle with UR corner folded down  
SPTR_FOLDER is 20 # File folder symbol  
SPTR_MULTIFILE is 21 # Multiple overlapping rectangles  
SPTR_PROGRAM is 22 # Rectangle, bold on top
```

Index

A

- ABS() 2-4
- accel.inc file 4-3
- action 1-3
- action bar 1-4
- action bar is 1-5
- Action Inquiry Built-in Functions
 - errorlevel 1-43, 1-126
 - found 1-43, 1-155
 - ioerror 1-43, 1-189
- action is 1-7
- Action Routine Definitions
 - action is 1-7
- Action Statements
 - action 1-3
 - activate 1-8
 - add 1-10
 - add item to 1-12
 - add to 1-14
 - add to class 1-17
 - add to item class 1-18
 - ancestry 1-19
 - append 1-21
 - begin 1-28
 - call 1-47
 - change 1-48
 - change action bar 1-50
 - change position 1-51
 - change priority 1-53
 - change size 1-54
 - change text 1-55
 - change to program 1-57
 - change window position 1-59
 - change window size 1-61
 - check 1-64
 - clear 1-76
 - close 1-239
 - copy 1-89
 - deemphasize 1-92
 - delete 1-93
 - delete action bar from 1-94
- Action Statements (*continued*)
 - delete from class 1-95
 - delete from item class 1-96
 - delete item 1-97
 - deselect 1-301
 - disable 1-119
 - emphasize 1-118
 - enable 1-119
 - exit 1-131
 - extract from 1-134
 - find 1-139
 - for each member loop 1-146
 - for each selected line
 - loop 1-148
 - for loop 1-150
 - highlight 1-167
 - invoke 1-188
 - leave block 1-203
 - leave loop 1-205
 - make 1-212
 - make block 1-213
 - make border invisible 1-216
 - make border visible 1-216
 - make cursor 1-218
 - make invisible 1-219
 - make parameter 1-220
 - make point disposition 1-221
 - make segment 1-213
 - make string disposition 1-223
 - make visible 1-219
 - open 1-239
 - plot 1-249
 - read 1-263
 - refresh 1-269
 - remove 1-270
 - resize 1-272
 - save program 1-295
 - select 1-301
 - send 1-305
 - set low memory threshold 1-311
 - set pointer to 1-312
 - squeeze memory 1-316

Action Statements (*continued*)

- start 1-317
- stop 1-325
- switch 1-330
- turn trace 1-346
- uncheck 1-64
- wait 1-357
- while loop 1-361
- write 1-369
- activate 1-8
- add 1-10
- add item to 1-12
- add to 1-14
- add to class 1-17
- add to item class 1-18
- ancestry 1-19, 1-40, 1-238
 - of objects 1-238
- ancestry of 1-20, 1-40
- apio 3-10
- append 1-21
- application 1-22
- ARCCOS() 2-5
- ARCSIN() 2-6
- ARCTAN2() 2-8
- ARCTAN() 2-7
- array 1-23
- Attribute Definitions
 - action bar 1-4
 - border 1-34
 - disabled 1-121
 - enabled 1-121
 - invisible 1-355
 - parameter is 1-243
 - priority is 1-254
 - visible 1-355

B

- b 3-3
- background at 1-26, 1-40, 1-152
- background of 1-27, 1-40
- BEEPFREQ 3-6
- begin 1-28
- blank block 1-31
- block 1-32

BLOCK Action Statement

- begin 1-28
- BOOPFREQ 3-6
- border 1-34
- border of 1-36, 1-40
- bottom [of] 1-42, 1-344
- box 1-38
- Built-in Functions 1-39
 - ancestry 1-40
 - ancestry of 1-40
 - background at 1-40
 - background of 1-40
 - border of 1-40
 - bottom [of] 1-42
 - checked in group 1-40
 - clipboard 1-43
 - column size of 1-40
 - date 1-43
 - errorlevel 1-43
 - eventnumber 1-40
 - eventparam 1-40
 - exists 1-41
 - foreground at 1-41
 - foreground of 1-41
 - found 1-43
 - freysize 1-43
 - handle of 1-41
 - input 1-40
 - ioerror 1-43
 - is checked 1-41, 1-43
 - is enabled 1-43
 - left [of] 1-41
 - length of 1-43
 - line size of 1-41
 - marker 1-43
 - member 1-43
 - members of 1-43
 - object 1-40
 - parameter 1-40
 - parameter of 1-41
 - priority of 1-41
 - right [of] 1-41
 - selectability of 1-41
 - selected line 1-44
 - selected line from 1-41
 - text of 1-41

Built-in Functions (*continued*)

- text of item 1-43
- textual line from 1-41
- time 1-44
- top [of] 1-42
- visibility of 1-42
- window xposition [of] 1-42
- window xsize [of] 1-42
- window yposition [of] 1-42
- window ysize [of] 1-42
- xcoord 1-40
- xcursor [of] 1-42
- xmiddle [of] 1-42
- xposition [of] 1-42
- xsize [of] 1-42
- ycoord 1-40
- ycursor [of] 1-42
- ymiddle [of] 1-42
- yposition [of] 1-42
- ysize [of] 1-42
- button 1-45

C

- call 1-47
- CEIL() 2-9
- change 1-48
- change action bar 1-50
- change position 1-51
- change priority 1-53
- change size 1-54
- change text 1-55
- change to program 1-57
- change window position 1-59
- change window size 1-61
- character set 1-63
- check 1-64
- check box 1-66
- checked 1-68
- checked in group 1-40, 1-69
- choice 1-70
- circle 1-73
- class 1-74
- clear 1-76
- clipboard 1-43, 1-78

- close 1-239
- CloseFile() 2-10
- CODEPAGE 3-6
- color 1-80
- COLOR Action Statements
 - make 1-212
 - make block 1-213
 - make border 1-215
 - make segment 1-213
- color values vii
- column size of 1-40, 1-83
- combination box 1-84
- command line options
 - apio 3-10
 - b 3-3
 - d 3-10
 - debug 3-3, 3-11
 - e 3-3, 3-11
 - f 3-4, 3-11
 - fullscreen 3-4
 - g 3-4, 3-11
 - lx 3-4
 - m 3-4, 3-11
 - nb 3-11
 - terp 3-11
- comments 1-87
- COMPILE.COMD OS/2 Command File 3-3
- Conditional Action Statement
 - if 1-170
- Configuration File
 - CONFIG.ESL 3-6
- configuration options
 - BEEPFREQ 3-6
 - BOOPFREQ 3-6
 - CODEPAGE 3-6
 - ESLERR 3-6
 - FontCNT 3-7
 - FontPATH 3-7
 - GLBMAX 3-7
 - GLBSIZE 3-7
 - HLDELAY 3-7
 - INCPATH 3-7
 - INCTYPE 3-7
 - KBDTABWID 3-7
 - LOCALDLL 3-7

configuration options (*continued*)

MODPATH 3-7
POINTER 3-8
PRGMAX 3-8
PRGSIZE 3-8
REMOTE 3-8
REMOTEDLL 3-8
SLEEPMAX 3-8
TABWIDTH 3-8
TEXTPATH 3-8
TEXTTYPE 3-8
XMAX 3-8
YMAX 3-8
CONFIG.ESL File 3-6
constant 1-88
copy 1-89
COS() 2-11

D

-d 3-10
date 1-43, 1-91
Date Library 2-2
 DaySpan() 2-12
 LocalDate 2-22
 LocalTime() 2-23
 Validdate() 2-54
 WeekDay() 2-57
datelib.inc file 4-4
DaySpan() 2-12
-debug 3-3, 3-11
Declarations
 function 1-158
 stimulus 1-324
 subroutine 1-326
deemphasize 1-92
Definitions
 array 1-23
 block 1-32
 button 1-45
 check box 1-66
 choice 1-70
 class 1-74
 combination box 1-84
 constant 1-88
 dialog box 1-99

Definitions (*continued*)

 dialog region 1-102
 dropdown list 1-113
 entry field 1-124
 fileid 1-138
 graphical region 1-159
 group box 1-164
 image region 1-172
 imagemap 1-178
 item class 1-194
 key 1-196
 list box 1-210
 multiline entry field 1-234
 pulldown 1-256
 push button 1-258
 radio button 1-261
 record 1-266
 segment 1-299
 sense region 1-307
 separator 1-310
 static text 1-322
 subroutine is 1-328
 textual region 1-338
 variable 1-352
delete 1-93
delete action bar from 1-94
delete from class 1-95
delete from item class 1-96
delete item 1-97
deselect 1-301
dialog box 1-99
dialog control objects 1-101
dialog region 1-102
disable 1-119
disabled 1-121
draw 1-108
draw arc 1-110
Drawing Statements
 blank block 1-31
 box 1-38
 circle 1-73
 draw 1-108
 draw arc 1-110
 ellipse 1-116
 insert 1-184
 move 1-229

Drawing Statements (*continued*)

- move arc 1-231
- overwrite 1-240
- polygon 1-251
- shape 1-313
- text 1-332
- wedge 1-359
- dropdown list 1-113

E

- e 3-3, 3-11
- EASEL OS/2 EE Library 2-2
 - EslQueryEgaColor() 2-14
 - EslQueryFocus() 2-15
 - EslSetEgaColor() 2-16
- EASELCMD OS/2 Command
 - File 3-10
- egacolor.inc include file 4-5
- ellipse 1-116
- emphasize 1-118
- enable 1-119
- enabled 1-121, 1-123
- entry field 1-124
- Environment Declaration
 - application 1-22
- Environment Declarations
 - font 1-140
 - precision 1-253
 - screen size 1-297
- errorlevel 1-43, 1-126
- escape sequences 1-127
- eslappc.inc include files 4-6
- eslaudio.inc include files 4-12
- ESLERR 3-6
- esllib.inc include files 4-14
- EslQueryEgaColor() 2-14
- EslQueryFocus() 2-15
- EslSetEgaColor() 2-16
- eventnumber 1-40, 1-291
- eventparam 1-40, 1-291
- exists 1-41, 1-130
- exit 1-131
- expressions 1-132
- EXP() 2-18

- extract from 1-134
 - SKIP Clause 1-134
 - TAKE Clause 1-134
- E() 2-13

F

- f 3-4, 3-11
- false 1-351
- File I/O Library 2-2
 - CloseFile() 2-10
 - GetError() 2-20
 - OpenFile() 2-34
 - ReadLineNumber() 2-38
 - ReadNextRecord() 2-40
 - ReadNext() 2-39
 - ReadRecordAtLine() 2-41
 - SetBufferSize() 2-44
 - SetIndexSize() 2-45
 - SetTabSize() 2-46
 - WriteRecord() 2-58
 - WriteString() 2-59
- fileid 1-138
- fileio.inc file 4-15
- find 1-139
- FLOOR() 2-19
- font 1-140
- Font Definitions
 - font is 1-143
- font is 1-143
- FONTCNT 3-7
- FONTPATH 3-7
- for each member loop 1-146
- for each selected line loop 1-148
- for loop 1-150
- foreground at 1-41, 1-152
- foreground of 1-41, 1-154
- found 1-43, 1-155
- frame attributes 1-156
- freesize 1-43, 1-157
- fullscreen 3-4
- function 1-158
- Functions
 - ABS() 2-4
 - ARCCOS() 2-5
 - ARCSIN() 2-6

Functions (*continued*)

- ARCTAN2() 2-8
- ARCTAN() 2-7
- CEIL() 2-9
- COS() 2-11
- DaySpan() 2-12
- EsiQueryEgaColor() 2-14
- EsiQueryFocus() 2-15
- EsiSetEgaColor() 2-16
- EXP() 2-18
- E() 2-13
- FLOOR() 2-19
- LN() 2-21
- LOG() 2-24
- MOD() 2-33
- PI() 2-36
- POWER() 2-37
- ReplyToMessage() 2-42
- SIN() 2-47
- SQRT() 2-50
- TAN() 2-53
- Validdate() 2-54
- WeekDay() 2-57

G

- g 3-4, 3-11
- GetError() 2-20
- GLBMAX 3-7
- GLBSIZE 3-7
- graphical objects 1-238
- graphical region 1-159
- group box 1-164

H

- handle of 1-41, 1-166
- highlight 1-167
- HLDELAY 3-7

I

- identifiers vii, 1-169
- if 1-170
- image region 1-172

- imagemap 1-178
- in 1-179
- include 1-181
- include files 4-1
 - accel.inc 4-3
 - datelib.inc 4-4
 - egacolor.inc 4-5
 - eslappc.inc 4-6
 - eslaudio.inc 4-12
 - esllib.inc 4-14
 - fileio.inc 4-15
 - mathlib.inc 4-16
 - message.inc 4-18
 - pmprtr.inc 4-20

- INCPATH 3-7

- INCTYPE 3-7

- input 1-40, 1-183

- insert 1-184

- invisible 1-355

- invoke 1-188

- ioerror 1-43, 1-189

- is checked 1-43, 1-190

- is enabled 1-43, 1-192

- item 1-193

- item class 1-194

- Item Inquiry Built-in Function 1-68

- checked 1-68

- Item Inquiry Built-in Functions

- enabled 1-123

- is checked 1-190

- is enabled 1-192

- text of item 1-335

K

- KBDTABWID 3-7

- key 1-196

- keyboard mapping table 1-198

- Keywords 1-202

- color 1-80

- in 1-179

- position 1-252

L

- leave block 1-203
- leave loop 1-205
- left [of] 1-41, 1-206
- length of 1-43, 1-208
- line size of 1-41, 1-209
- list box 1-210
- LN() 2-21
- LocalDate() 2-22
- LOCALDLL 3-7
- LocalTime() 2-23
- LOG() 2-24
- Loop Action Statements
 - for each member loop 1-146
 - for each selected line
 - loop 1-148
 - for loop 1-150
 - while loop 1-361
- lx 3-4

M

- m 3-4, 3-11
- make 1-212
- make block 1-213
- make border 1-215
- make border invisible 1-216
- make border visible 1-216
- make cursor 1-218
- make invisible 1-219
- make parameter 1-220
- make point disposition 1-221
- make segment 1-213
- make string disposition 1-223
- make visible 1-219
- marker 1-43, 1-134
- MATCH_CLAUSE 1-279, 1-330
- Math Library 2-3
 - ABS() 2-4
 - ARCCOS() 2-5
 - ARCSIN() 2-6
 - ARCTAN2() 2-8
 - ARCTAN() 2-7
 - CEIL() 2-9
 - COS() 2-11

Math Library (continued)

- EXP() 2-18
- E() 2-13
- FLOOR() 2-19
- LN() 2-21
- LOG() 2-24
- MAXINT() 2-26
- MAX() 2-25
- MEANINT() 2-28
- MEAN() 2-27
- MEDIANINT() 2-30
- MEDIAN() 2-29
- MININT() 2-32
- MIN() 2-31
- MOD() 2-33
- PI() 2-36
- POWER() 2-37
- SIN() 2-47
- SQRT() 2-50
- STDDEVINT() Subroutine 2-49
- STDDEV() 2-48
- SUMINT() 2-52
- SUM() 2-51
- TAN() 2-53
- VARIANCEINT() 2-56
- VARIANCE() 2-55
- mathlib.inc file 4-16
- MAXINT() 2-26
- MAX() 2-25
- MEANINT() 2-28
- MEAN() 2-27
- MEDIANINT() 2-30
- MEDIAN() 2-29
- member 1-43, 1-226
- members of 1-43, 1-227
- Message Library 2-3
 - ReplyToMessage() 2-42
- message.inc file 4-18
- MININT() 2-32
- MIN() 2-31
- MODPATH 3-7
- module 1-228
- MOD() 2-33
- move 1-229
- move arc 1-231

multiline entry field 1-234

N

-nb 3-11

notation conventions v

O

object 1-40, 1-237, 1-238

 Ancestry 1-238

 Types 1-238

Object Inquiry Built-in

 Function 1-68

 checked 1-68

Object Inquiry Built-in Functions

 ancestry of 1-20

 background at 1-26, 1-152

 background of 1-27

 border of 1-36

 bottom [of] 1-344

 checked in group 1-69

 column size of 1-83

 exists 1-130

 foreground at 1-152

 foreground of 1-154

 handle of 1-166

 is checked 1-190

 left [of] 1-206

 line size of 1-209

 parameter of 1-244

 priority of 1-255

 right [of] 1-206

 selectability of 1-302

 text of 1-334

 textual 1-336

 top [of] 1-344

 visibility 1-354

 window xposition [of] 1-364

 window xsize [of] 1-366

 window yposition [of] 1-364

 window ysize [of] 1-366

 xcursor [of] 1-372

 xmiddle [of] 1-373

 xposition [of] 1-374

 xsize [of] 1-375

Object Inquiry Built-in Functions

(continued)

 ycursor [of] 1-372

 ymiddle [of] 1-373

 yposition [of] 1-374

 ysize [of] 1-375

Object Response Definitions

 response 1-274

open 1-239

OpenFile() 2-34

operands 1-132

operators 1-132

OS/2 Command Files

 COMPILE.CMD 3-3

 EASELCMD 3-10

overwrite 1-240

P

parameter 1-40, 1-242

parameter is 1-243

parameter of 1-41, 1-244

parent objects 1-238

pattern 1-245

Pattern Definitions

 pattern is 1-248

pattern is 1-248

Pattern References

 pattern 1-245

PI() 2-36

plot 1-249

pmptr.inc file 4-20

POINTER 3-8

polygon 1-251

position 1-252

POWER() 2-37

precision 1-253

PRGMAX 3-8

PRGSIZE 3-8

Printing Utilities

 priority is 1-254

 priority of 1-41, 1-255

 pulldown 1-256

 push button 1-258

R

- radio 1-261
- read 1-263
- ReadLineNumber() 2-38
- ReadNextRecord() 2-40
- ReadNext() 2-39
- ReadRecordAtLine() 2-41
- record 1-266
- References
 - array 1-23
 - include 1-181
 - item 1-193
 - module 1-228
- refresh 1-269
- REMOTE 3-8
- REMOTEDLL 3-8
- remove 1-270
- ReplyToMessage() 2-42
- resize 1-272
- Response Definitions
 - response to char 1-279
 - response to interval 1-283
 - response to item 1-284
 - response to line 1-279
 - response to low memory 1-287
 - response to start 1-289
 - response to stimulus 1-290
 - response to termination 1-292
 - response to timeout 1-293
- Response Inquiry Built-in Function
 - eventnumber 1-291
 - eventparam 1-291
- Response Inquiry Built-in Functions
 - input 1-183
 - object 1-237
 - parameter 1-242
 - xcoord 1-371
 - ycoord 1-371
- response to 1-274
- response to char 1-279
 - MATCH Clause 1-279
- response to interval 1-283
- response to item 1-284
- response to line 1-279
 - MATCH Clause 1-279

- response to low memory 1-287
- response to start 1-289
- response to stimulus 1-290
- response to termination 1-292
- response to timeout 1-293
- right [of] 1-41, 1-206

S

- save program 1-295
- screen size 1-297
- segment 1-299
- select 1-301
- selectability of 1-41, 1-302
- selected line 1-44, 1-303
- selected line from 1-41, 1-304
- send 1-305
- sense region 1-307
- separator 1-310
- set low memory threshold 1-311
- set pointer to 1-312
- SetBufferSize() 2-44
- SetIndexSize() 2-45
- SetTabSize() 2-46
- shape 1-313
- SIN() 2-47
- SKIP Clause 1-134
- SLEEPMAX 3-8
- Special Built-in Function
 - marker 1-134
- Special Built-in Functions
 - clipboard 1-78
- Special Inquiry Built-in Functions
 - date 1-91
 - free size 1-157
 - length of 1-208
 - member 1-226
 - members 1-227
 - selected line 1-303
 - selected line from 1-304
 - time 1-343
- SQRT() 2-50
- squeeze memory 1-316
- start 1-317
- static text 1-322

STDDEVINT() 2-49
 STDDEV() 2-48
 stimulus 1-324
 stop 1-325
 string
 escape sequences 1-127
 subroutine 1-326
 subroutine is 1-328
 Subroutines
 CloseFile() 2-10
 GetError() 2-20
 MAXINT() 2-26
 MAX() 2-25
 MEANINT() 2-28
 MEAN() 2-27
 MEDIANINT() 2-30
 MEDIAN() 2-29
 MININT() 2-32
 MIN() 2-31
 OpenFile() 2-34
 ReadLineNumber() 2-38
 ReadNextRecord() 2-40
 ReadNext() 2-39
 ReadRecordAtLine() 2-41
 SetBufferSize() 2-44
 SetIndexSize() 2-45
 SetTabSize() 2-46
 STDDEVINT() 2-49
 STDDEV() 2-48
 SUMINT() 2-52
 SUM() 2-51
 VARIANCEINT() 2-56
 VARIANCE() 2-55
 WriteRecord() 2-58
 WriteString() 2-59
 SUMINT() 2-52
 SUM() 2-51
 switch 1-330
 MATCH_CLAUSE 1-330

T

TABWIDTH 3-8
 TAKE_CLAUSE 1-134
 TAN() 2-53

Template Definitions
 action bar is 1-5
 -terp 3-11
 text 1-332
 text of 1-41, 1-334
 text of item 1-43, 1-335
 TEXTPATH 3-8
 TEXTTYPE 3-8
 textual 1-336
 textual line from 1-41
 textual region 1-338
 time 1-44, 1-343
 top [of] 1-42, 1-344
 true 1-351
 turn trace 1-346
 type conversions 1-347

U

unchecked 1-64

V

Validate() 2-54
 values 1-350
 variable 1-352
 VARIANCEINT() 2-56
 VARIANCE() 2-55
 visibility of 1-42, 1-354
 visible 1-355

W

wait 1-357
 wedge 1-359
 WeekDay() 2-57
 while loop 1-361
 window xposition [of] 1-42, 1-364
 window xsize [of] 1-42, 1-366
 window yposition [of] 1-42, 1-364
 window ysize [of] 1-42, 1-366
 write 1-369
 WriteRecord() 2-58
 WriteString() 2-59

X

xcoord 1-40

xcoord Response 1-371

xcursor [of] 1-42, 1-372

XMAX 3-8

xmiddle [of] 1-42, 1-373

xposition [of] 1-42, 1-374

xsize [of] 1-42, 1-375

Y

ycoord 1-40

ycoord Response 1-371

ycursor [of] 1-42, 1-372

YMAX 3-8

ymiddle [of] 1-42, 1-373

yposition [of] 1-42, 1-374

ysize [of] 1-42, 1-375

EASEL Version 1.1 for OS/2 Extended Edition

Language Reference Manual

SC38-7035-01

Reader's Comment Form

Please use this form only to identify publication errors or to request changes in publications. Direct any requests for additional publications, or technical information about IBM systems, to your IBM representative or nearest IBM branch office.

You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Comments:

If you wish a reply, give your name, company, mailing address, and date.

Name _____
Company _____
Address _____
City _____ State _____ Zip Code _____

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

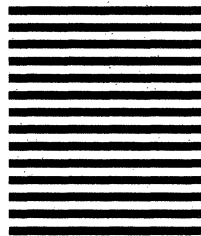


**NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES**

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40

ARMONK, NEW YORK 10504



POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
10401 Fernwood Road
Bethesda, MD 20817

ATTENTION: Software Development, Dept. 877

.....
Fold Here

Tape

Please Do Not Staple

Tape

Continued from inside front cover

SUCH WARRANTIES ARE IN LIEU OF ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow the exclusion of implied warranties, so the above exclusion may not apply to you.

LIMITATION OF REMEDIES

IBM's entire liability and your exclusive remedy shall be as follows:

1. IBM will provide the warranty described in IBM's Statement of Limited Warranty. If IBM does not replace defective media or, if applicable, make the Program operate as warranted or replace the Program with a functionally equivalent Program, all as warranted you may terminate your license and your money will be refunded upon the return of all of your copies of the Program.
2. For any claim arising out of IBM's limited warranty, or for any other claim whatsoever related to the subject matter of this Agreement, IBM's liability for actual damages, regardless of the form of action, shall be limited to the greater of \$5,000 or the money paid to IBM, its Authorized Dealer or its approved supplier for the license for the Program that caused the damages or that is the subject matter of, or is directly related to, the cause of action. This limitation will not apply to claims for personal injury or damages to real or tangible personal property caused by IBM's negligence.
3. In no event will IBM be liable for any lost profits, lost savings, or any incidental damages or other conse-

quential damages, even if IBM, its Authorized Dealer or its approved supplier has been advised of the possibility of such damages, or for any claim by you based on a third party claim.

Some states do not allow the limitation or exclusion of incidental or consequential damages so the above limitation or exclusion may not apply to you.

GENERAL

You may terminate your license at any time by destroying all your copies of the Program or as otherwise described in this Agreement.

IBM may terminate your license if you fail to comply with the terms and conditions of this Agreement. Upon such termination you agree to destroy all your copies of the Program.

Any attempt to sublicense, rent, lease or assign, or, except as expressly provided herein, to transfer any copy of the Program is void.

You agree that you are responsible for payment of any taxes, including personal property taxes, resulting from this Agreement.

No action, regardless of form, arising out of this Agreement may be brought by either party more than two years after the cause of action has arisen except for breach of the provisions in the Section entitled "License" in which event four years shall apply.

This Agreement will be construed under the Uniform Commercial Code of the State of New York.

Z125-3301-02 4/87
Printed in U.S.A.



SC38-7035-01



Printed in U.S.A.